Sequana

Release 0.14.0

Thomas Cokelaer

Mar 30, 2022
## CONTENTS

1 SEQUANA
   1.1 Contributors .......................................................... 4
   1.2 What is Sequana ? ...................................................... 5

2 User guide and reference ................................. 11
   2.1 Installation ............................................................... 11
   2.2 Overview ................................................................. 16
   2.3 Tutorial ................................................................. 20
   2.4 Pipelines ................................................................. 27
   2.5 Gallery ................................................................. 29
   2.6 Case Examples ......................................................... 39
   2.7 Applications (standalone) ....................................... 43
   2.8 Sequanix Tutorial ................................................... 47
   2.9 Developer guide ...................................................... 65
   2.10 Rules ................................................................. 78
   2.11 Wrappers ............................................................... 78
   2.12 References ........................................................... 79
   2.13 References (Viz) .................................................... 166
   2.14 References (stats) ................................................... 187
   2.15 FAQs ................................................................. 193
   2.16 Changelog .............................................................. 199
   2.17 Glossary ............................................................... 219
   2.18 Indices and tables ............................................... 220

Python Module Index ........................................ 221

Index ........................................................................ 223
Python version 3.7, 3.8, 3.9

Documentation On readthedocs

Issues On github

How to cite Citations are important for us to carry on developments. For Sequana library (including the pipelines), please use


For the genome coverage tool (sequana_coverage): Dimitri Desvillechabrol, Christiane Bouchier, Sean Kennedy, Thomas Cokelaer http://biorxiv.org/content/early/2016/12/08/092478


Sequana includes a set of pipelines related to NGS (new generation sequencing) including quality control, variant calling, coverage, taxonomy, transcriptomics. We also ship Sequanix, a graphical user interface for Snakemake pipelines.
Table 1: Pipelines and tools available in the Sequana project

<table>
<thead>
<tr>
<th>pipeline or tools</th>
<th>Latest Pypi version</th>
<th>Test passing</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://github.com/sequana/sequana_pipetools">https://github.com/sequana/sequana_pipetools</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="https://github.com/sequana/sequana-wrappers">https://github.com/sequana/sequana-wrappers</a></td>
<td>Not on pypi</td>
<td></td>
</tr>
<tr>
<td><a href="https://github.com/sequana/demultiplex">https://github.com/sequana/demultiplex</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="https://github.com/sequana/fastqc">https://github.com/sequana/fastqc</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="https://github.com/sequana/mapper">https://github.com/sequana/mapper</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="https://github.com/sequana/pacbio_qc">https://github.com/sequana/pacbio_qc</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="https://github.com/sequana/ribofinder">https://github.com/sequana/ribofinder</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="https://github.com/sequana/rnaseq">https://github.com/sequana/rnaseq</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="https://github.com/sequana/variant_calling">https://github.com/sequana/variant_calling</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please see the documentation for an up-to-date status and documentation.

1.1 Contributors

Maintaining BioServices would not have been possible without users and contributors. Each contribution has been an encouragement to pursue this project. Thanks to all:
# 1.1.1 Changelog

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0.14.0  | • pinned click>=8.1.0 due to API change (autocomplete)  
|         | • moved tests around to decrease packaging from 16 to 4Mb  
|         | • ribodesigner: new plots, clustering and notebook |
| 0.13.X  | • Remove useless standalones or moved to main sequana command  
|         | • Move sequana_lane_merging into a subcommand (sequana lane_merging)  
|         | • General cleanup of documentation, test and links to pipelines  
|         | • add new ribodesigner subcommand |
| 0.12.7  | • Fix memory leak in len() of FastA class |
| 0.12.6  | • remove some rules now in https://github.com/sequana/sequana-wrappers |
| 0.12.5  | • refactorisation of VCF tools/modules to use vcfpy instead of pyVCF |
| 0.12.4  | • complete change log before 0.12.4 on readthedocs.org |

## 1.2 What is Sequana?

Sequana is a versatile tool that provides

1. A Python library dedicated to NGS analysis (e.g., tools to visualise standard NGS formats).
2. A set of pipelines dedicated to NGS in the form of Snakefiles (Makefile-like with Python syntax based on snakemake framework) with more than 80 re-usable rules (see Rules).
3. Original tools to help in the creation of such pipelines including HTML reports.
4. **Standalone applications:**
   1. sequana_coverage ease the extraction of genomic regions of interest and genome coverage information
   2. sequana_taxonomy performs a quick taxonomy of your FastQ. This requires dedicated databases to be downloaded.
   3. Sequanix: GUI for snakemake workflows, a GUI for Snakemake workflows (hence Sequana pipelines as well)

The sequana pipelines are various. Since March 2020, they have their own independent life within dedicated github repositories. You may find pipelines for NGS quality control (e.g. adapters removal, phix removal, trimming of bad
quality bases), variant calling, characterisation of the genome coverage, taxonomic classification, de-novo assembly,
*Variant calling*, RNA-seq, etc. See the *Pipelines* section for more information.

**Sequana** can be used by developers to create new pipelines and by users in the form of applications ready for production. Moreover, **SequaniX** can be used to set the parameters of pipelines and execute them easily with a graphical user interface.

To join the project, please let us know on [github](https://github).
1.2. What is Sequana?
2.1 Installation

Here below are the instructions to install Sequana. There are different ways (source, bioconda, singularity, conda environment, pip). Let us summarize the different methods for you.

If you want the latest version of Sequana, you should install it from source (see From GitHub Source code). Otherwise, you can install a release of Sequana from the Pypi website (using pip). Note that for pipelines, which are now independent Python packages, we also use Pypi releases. However, third-party dependencies (not Python) should be installed manually. Most of them are provided through Anaconda channels. See the From bioconda Section for details on how to set up Conda.

For instance, if you want to use the sequana_fastqc pipelinem you must install fastqc yourself, which is not a Python package.

If you just want to test Sequana or Sequanix (see note here below) or one of the Sequana standalone, we also provide Singularity containers as explained in the Singularity section.

**Design choice**

Since version 0.8.0, we decided to move the pipelines outside of the main sequana library. This choice was made to face the increase of pipelines available in the Sequana project. Indeed, each pipeline comes with its own dependencies, which are not necessarily Python. The full installation of Sequana started to be cumbersome even for experienced users. We dealt with this issue using bioconda. Yet, even with such solutions it started to be difficult to manage easy installation. So, as usual, divide and conquer: each pipeline has now its own life cycle outside of Sequana. For example, the variant calling pipeline is hosted on https://github.com/sequana/variant_calling. This way, you can install Sequana quite easily using pip.

**Sequanix**

Sequanix has now its own repository here: https://github.com/sequana/sequanix and should be installed independently.
2.1.1 Latest recommended installation method

Sequana is maintained under Python 3.7 and above (Feb 2022).

**Warning:** For Sequanix, for a while we advised to use Python 3.7.3 due to a PyQt library issue preventing Sequanix to work.

Lots of dependencies have been dropped in version 0.8.0 so that you could simply use `pip` to install Sequana.

In any case we strongly recommend to use a virtual environment so that (i) you can install all requirements without root permissions and (ii) you do not interfere with your system.

We will use `conda` for that. Before starting you should install and set the channels as explained in the *From bioconda* section. Then, create an environment:

```
conda create --name sequana_env python=3.7.3
source activate sequana_env
```

### pip installation

For the latest release of Sequana:

```
pip install sequana --upgrade
```

This will install all Python dependencies such as Pandas, Numpy, etc. It will take about 5-10 minutes to install this version.

**Note:** If you want to use Sequanix, which rely on PyQt5, please install PyQt5 using conda:

```
conda install -c anaconda qt pyqt>5
```

Using pip may lead to compatibility issues with your underlying Qt library, which must be available to install PyQt. PyQt5, v5.9.2 is known to work. v5.15.0 fails (PyQt5.QtWebEngineWidgets)

With Python 3.6.12, this is compatible:

```
pip install "pyqt5<=5.10"
```

### pipelines

Sequana pipelines are now easily installable using `pip`:

```
pip install sequana_rnaseq
pip install sequana_fastqc
pip install sequana_demultiplex
pip install sequana_pacbio_qc
# etc
```

The dependencies of this pipeline must be dealt with by the developer or users. Each pipeline has its own repository on github (https://github.com/sequana/sequana_PIPELINENAME) where more details about specific dependencies are provided.
A set of predefined pipelines can be installed using:

```
pip install sequana[pipelines]
```

### 2.1.2 Other solutions (overview)

1. **Singularity** (tested with version 2.4.2; see below for installation). Strictly speaking, there is no compilation. This method is for testing and production. It downloads an image/container that is ready-to-use (here the latest available release):

```
# NOTE THAT THIS IS AN OLD RELEASE 0.6.5
singularity pull --name sequana.img shub://sequana/sequana
```

and can be used as follows (for example):

```
singularity exec sequana.img sequanix --help
```

See *Singularity* section to install a specific release and more details.

2. **Bioconda.** *Sequana* is available on conda/bioconda as a pre-compiled package:

```
# Note that its version may be behind the pypi releases
conda install sequana
```

3. From source. If you prefer to install everything yourself, the source code is available on github ([http://github.com/sequana/sequana](http://github.com/sequana/sequana)):

```
git clone https://github.com/sequana/sequana
cd sequana
pip install sequana
```

These three methods are detailed hereafter.

### 2.1.3 From bioconda

If you have not installed *Sequana*, be aware that many dependencies need to be compiled (i.e., time consuming and requires proper C compiler). Besides, many pipelines rely on third-party software such as BWA or samtools that are not Python libraries. We therefore recommend to use *conda* that provides pre-compiled software for you.

**Install conda executable**

**Warning:** this is currently broken on bioconda. We advise you to install sequana with Python (pip) for the latest versions.

In practice, we do use *Anaconda*. We recommend to install *conda* executable via the manual installer ([download](http://anaconda)). You may have the choice between Python 2 and 3. We recommend to choose a Python version 3.
Add bioconda channels

When you want to install a new package, you have to use this type of syntax:

```bash
conda install ipython
```

where `ipython` is the package you wish to install. Note that by default, `conda` looks on the official Anaconda website (channel). However, there are many channels available. We will use the `bioconda` channel. To use it, type these commands (once for all):

```bash
conda config --add channels r
conda config --add channels defaults
conda config --add channels conda-forge
conda config --add channels bioconda
```

**Warning:** it is important to add them in this order, as mentioned on bioconda webpage (https://bioconda.github.io/).

If you have already set the channels, please check that the order is correct. With the following command:

```bash
conda config --get channels
```

You should see:

```
--add channels 'r'  # lowest priority
--add channels 'defaults'
--add channels 'conda-forge'
--add channels 'bioconda'  # highest priority
```

As of May 2020, the recommended order is now:

```bash
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
```

Create an environment

Once `conda` is installed and the channels set, open a new shell. Although this is not required strictly speaking, we would recommend to create an environment dedicated to Sequana. This environment can later be removed without affecting your system or conda installation. A `conda` environment is nothing else than a directory and can be created as follows:

```bash
conda create --name sequana_env python=3.7.3
```

Then, since you may have several environments, you must activate the `sequana` environment itself (each time you open a new shell):

```bash
source activate sequana_env
```
Installation

Sequana is on bioconda. You can follow these instructions or type:

```bash
conda install sequana
```

### 2.1.4 From GitHub Source code

Finally, if you are a developer and wish to use the latest code, you can install `sequana` in develop mode as follows:

```bash
conda create --name sequana python=3.7.3
source activate sequana
git clone git@https://github.com:sequana/sequana.git
cd sequana
pip install -e .

# to perform testing and documentation:
pip install -e .[doc,testing]
```

This should install most of the required dependencies. However, you may need to install more packages depending on the pipeline used (related to Qt for instance).

### 2.1.5 Singularity

**Warning:** this is now up-to-date. Come back later or contribute to this section.

We provide Singularity images on https://singularity-hub.org/collections/114/. They contain Sequana standalones and some of the pipelines dependencies as well as Sequanix. Note, however, that Sequanix relies on PyQt (graphical environment) and would work for Linux users only for the time being. The main reason being that under Mac and windows a virtualbox is used by Singularity preventing a X connection.

First, install singularity (http://singularity.lbl.gov/). You must use at least version 3.5. We suggest users to look at the l=singularity installation page itself to install the tool.

Once done, you can either build an image yourself or download a Sequana image. For instance, for the latest master version:

```bash
singularity pull --name sequana.img shub://sequana/sequana:latest
```

or for the release 0.6.3:

```bash
singularity pull --name sequana_0_6_3.img shub://sequana/sequana:0_6_3
```

The term latest in Singularity Hub will pull, across all of your branches and tags, the most recent image, so if you come back in a year and get the latest (or ommit tha tag), you may not get the same container ! So, it is best using a specific tag.

Do not interrupt the download (1.5Go). Once downloaded, you can use, for instance, the sequana_coverage executable:

```bash
singularity exec sequana.img sequana_coverage --help
```

or sequanix:
Would you miss a dependency, just enter into the singularity container and install the missing dependencies. You will need writable permission:

```
sudo singularity shell -w sequana.img
```

Then, inside the container, install or fix the problem and type exit to save the container.

**Note:** you may need to install squashfs-tools (e.g. yum install squashfs-tools)

## 2.2 Overview

**Contents**

- **Overview**
  - *Sequana is a Python library*
    - Example 1: running median on coverage
    - Example2: read a fastq file
  - *Sequana provides standalone applications*
  - *Sequana, a pipeline construction facilitator*
    - Installation of a pipeline
    - Usage
    - Using Sequanix standalone
  - *Sequana Reports*

Sequana provides standalone applications (e.g., sequana_coverage, sequana_taxonomy) and pipelines in the form of Snakefiles. Although the standalone applications are usually simpler, they may not have all features or parameters offered by the pipelines. Since version 0.8.0, most pipelines have been moved to different repository with one repository per pipeline. This was done to make pipelines indepence and the Sequana more modular and effective for deployment in production mode.

The Tutorial, Pipelines, Case Examples sections provide many examples on their usage. Check also the Gallery section for code snippets.

This section will not describe all available standalones and pipelines. We will focus on one example (coverage) to show how one can use the Sequana library, or standalone application, or pipeline to get information about the coverage of a set of mapped reads onto a reference.
2.2.1 Sequana is a Python library

Example 1: running median on coverage

Sequana is a Python library. It contains many functionalities, which are fully documented and available in the References (stats) section. We can first look at the coverage contained within a BED file using the library. First, we need some data. Sequana provides some test examples, which can be accessed using sequana_data() function. The test case is a virus (about 18,000 bases):

```python
from sequana import sequana_data
filename = sequana_data('JB409847.bed')
```

We can then use the GenomeCov class to read the file:

```python
from sequana import GenomeCov
gc = GenomeCov(filename)
```

Select a chromosome (first one) and compute the running median:

```python
chrom = gc[0]
chrom.running_median(n=5001, circular=True)
chrom.compute_zscore()
```

and finally plot the coverage together with confidence interval (3 sigma):

```python
chrom.plot_coverage()
```

See also:

notebook available in the github repository

As you can see, Sequana is a standard Python library where developers can select functions, classes, modules to help them design new tools and pipelines.

Example2: read a fastq file

Let us use the FastQC class to get the distribution of the bases ACGT across all reads of a FastQ file.

```python
from sequana import FastQC
from sequana import sequana_data
filename = sequana_data("test.fastq")

fastqc = FastQC(filename)
print(fastqc.fastq)
for x in 'ACGT':
    fastqc.get_actg_content()[x].hist(alpha=0.5, label=x, histtype='step', lw=3, bins=10)

from pylab import legend
legend()
```

Many more functionalities are available. The reference guide should help you.
2.2.2 Sequana provides standalone applications

The Python example about the coverage is actually quite useful. We therefore decided to provide a standalone application. There are other standalone applications listed in Applications (standalone) section.

The one related to the coverage example shown above is named sequana_coverage. If you have a BED file, type:

```
sequana_coverage -i <BEDFILENAME>
```

If your organism has a circular DNA, add -o. You can play with the window size for the running median using -w.

Using the BED file and reference mentioned in the previous section you should obtain the same figure as above.

An additional feature is the report using --show-html option.

2.2.3 Sequana, a pipeline construction facilitator

In Sequana, in addition to the library and standalone applications, we also provide a set of pipelines (see Pipelines section). Originally, pipeline were provided with Sequana, inside the same source repository. Since version 0.8.0, pipeline have their own repository. For instance, Variant Calling is available on https://github.com/sequana/variant_calling. We will not describe all pipelines here below since new ones may appear now and then. Instead, let us explain the way pipelines can be designed and run.

Installation of a pipeline

With the new design implemented in v0.8.0, pipelines are independent Python packages posted on Pypi. You can now install a pipeline (e.g., variant calling) as follows in your virtual environment:

```
pip install sequana_variant_calling --upgrade
```

The --upgrade option is to make sure you install the newest version.

To check if the installation is successful, just type:

```
sequana_variant_calling --help
```

Usage

A very simple and useful pipeline for this explanation is the sequana_fastqc pipeline. Install it as follows:

```
pip install sequana_fastqc
```

and check the help message:

```
sequana_fastqc --help
```

You will see 4 sections some of which are common to all Sequana pipelines.

The generic section allows use to print the help with --help, to set the level of information printed to the screen (level), the version (version). Pipelines can be run locally or on a SLURM clusters. This can be set with the --run-mode option. Note, however, that this option is set automatically to slurm-mode if slurm commands are found (e.g. sbatch).

The slurm section can be used to set slurm options for Snakemake. If you do not know what it means, let it be the defaults values. Just note that memory usage is set to 4Gb by default and number of cores is limited to 4 per job.
The *snakemake* section allows you to set to maximum number of jobs to be used, which is set to 4 (if run-mode is set to local) and 40 (if run-mode is set to slurm).

The --working-directory is set to the name of the pipeline and is the important parameter. It tells sequana where to store the pipeline files (e.g., snakemake, configuration files). You can change it to your will but if it exists already, the pipeline will not be set up and you will need to use the --force option to overwrite existing files.

The next section is about your input data. Most of the pipelines expect to find Illumina data with single or paired-end data sets. The directory where to find the data is defined by the --input-directory parameter. You can refine the search by providing an input pattern, which is set to *fastq.gz* by default. Since, Illumina data may be paired, we have a mechanism to check and discovered paired data for each sample. By default, the paired data are differentiate thanks to a pattern _R1_ or _R2_ to be found in the filenames. The common pattern set with --input-readtag is set to _R[12]_ but can be easily changed. For instance if your files do not contain the R or if the _R1 is to be found at the end of the file, just change it accordingly.

So, let us now perform the fastqc of a bunch of samples. You could type:

```
sequana_fastqc --input-directory my_data_directory --working-directory test1
```

This will copy the snakefile, the configuration files and useful files to run the analysis. Follow the instructions that is:

```
cd test1
```

In this directory, you can find The configuration file called *config.yaml*. This pipeline is very simple but you can see the parameters related to your input data:

```
input_directory: /home/login_example/data_example/my_data_directory
input_readtag: _R[12]_
input_pattern: '*fastq.gz'
```

So you can edit this file to correct it or change other parameters. If you are happy with those choices, it is now time to run the pipeline. If you know snakemake, you can just use it. For example:

```
snakemake -s fastqc.rules
```

or just type:

```
sh fastq.sh
```

Wait and see. Once done. If every went well, you can keep the configuration files and pipeline-related files, or delete them using:

```
make clean
```

See also:

*Pipelines* section for more information.
Using Sequanix standalone

An even easier way is to use our graphical interface named **Sequanix**. A snapshot can be found in the *Sequanix: GUI for snakemake workflows* section and a tutorial in *Sequanix Tutorial*.

Note, however, that the Sequanix interface is slightly different. The content of the working directory may differ slightly for the time being. The advantage of using Sequanix is that complex configuration pipeline can be tuned easily through its graphical interface.

### 2.2.4 Sequana Reports

Pipelines and standalone make use of internal reporting. Since they are part of the **Sequana** library, they can also be used with your own code. For instance, if you have a BAM file, you can use the following code to create a basic report:

```python
from sequana import BAM, sequana_data
from sequana.modules_report.bamqc import BAMQCModule
filename = sequana_data("test.bam", "testing")
r = BAMQCModule(filename, "bam.html")
```

that results can be shown in `bam.html`

### 2.3 Tutorial

Here below you can find tutorials on how to use some of the sequana pipelines and standalones. This gives you a flavor of what can be achieved using **Sequana**. Dedicated sections on pipelines and applications are found in other pages and can be a complement to this quick overview.

**Contents**

- **Tutorial**
  - The standalone Sequana
  - The fastqc pipeline
  - Quality Control pipelines
  - Taxonomy (standalone)
  - Variant calling pipeline
  - De novo
  - RNA-seq
  - Singularity and Sequanix
2.3.1 The standalone Sequana

New since version 0.9.0. We are a single entry point for a set of tools used in pipelines or as standalone applications. You can type:

```
sequana --help
```

to get the list of applications. Would you need completion, this is possible using e.g. for bash users:

```
eval "${_SEQUANA_COMPLETE=source_bash sequana}"
```

To speed up things, you can also save the script somewhere:

```
_FOO_BAR_COMPLETE=bash_source foo-bar > ~/.foo-bar-complete.bash
```

and then source the file in ~/.bashrc:

```
. ~/.foo-bar-complete.bash
```

See also:


2.3.2 The fastqc pipeline

The following example will show how to run the fastqc pipeline ([https://github.com/sequana/fastqc](https://github.com/sequana/fastqc)) on a pair of FastQ files. The data comes from a sequencing (using HiSeq technology) of a Measles virus. For testing purposes, you can download R1 and R2 files that contain only 1500 reads. Copy them in a local directory.

Those files are from an HiSeq2500 run. The adapters are PCRFree. There is only one sample for which the index is GTGAAA. You should have 10% of adapters.

Then, initiate the pipeline:

```
sequana_fastqc --input-directory .
cd fastqc
sh fastq.sh
```

Open the summary.html file that is generated for you.

2.3.3 Quality Control pipelines

The quality_control pipeline ([https://github.com/sequana/quality_control](https://github.com/sequana/quality_control)) is not maintained anymore and has been split into several smaller pipelines.

For book-keeping, we keep this section though.

The following example will show how to run the quality control pipeline ([https://github.com/sequana/quality_control](https://github.com/sequana/quality_control)) on a pair of FastQ files. The data comes from a sequencing (using HiSeq technology) of a Measles virus. For testing purposes, you can download R1 and R2 files that contain only 1500 reads. Copy them in a local directory.

Those files are from an HiSeq2500 run. The adapters are PCRFree. There is only one sample for which the index is GTGAAA. You should have 10% of adapters.

Make sure you have installed the pipeline:
This example shows how to initialise and run the quality control pipeline on a pair of FastQ files. Copy the two data files (link above) into the local directory where you will initiate the pipeline.

First, run the sequana standalone application to initialise the pipeline `quality_control`:

```
sequana_quality_control --cutadapt-adapter-choice TruSeq
```

This command fills the required configuration file(s) and copy it along the pipeline itself inside the default working directory (`quality_control`).

The pipeline does 3 things:

1. remove the Phix if present
2. apply cutadapt to trim the bases with quality below 30 and removes adapters (here TruSeq)
3. taxonomy if Kraken databases are provided.

By default, the output directory is called `quality_control` and can be overwritten with the `--working-directory` parameter. Then, run the pipeline and wait for completion:

```
cd quality_control
snakemake -s quality_control.rules --stats stats.txt -p -j 4 --forceall
```

The `-p` option shows the commands, `-j 4` means use 4 threads when possible. Alternatively, there is also a `runme.sh` script.

---

**Note:** you can also use the shell script `sh quality_control.sh` instead of the snakemake command.

You should now have a directory with a HTML report corresponding to the sample:

```
open index.html
```

### 2.3.4 Taxonomy (standalone)

To perform a quick taxonomy of your reads, you can use `sequana_taxonomy` either from Python or as a standalone. Here we show how to use the Python approach (see Applications (standalone)) for the other approach.

Download a toy kraken database designed for this problem (contains only 100 FASTA files mixing measles viruses and others viruses):

```
from sequana import KrakenDownload, sequana_config_path
kd = KrakenDownload()
kd.download("toydb")
database_path = sequana_config_path + "/kraken_toydb"
```
Then, you may use the following code to perform the analysis (using `sequana.kraken`):

```python
from sequana import KrakenPipeline
kp = KrakenPipeline(\["R1.fastq.gz", "R2.fastq.gz"], database="~/.config/sequana/kraken_toydb")
kp.run()
```

Alternatively, you can use the standalone application:

```bash
sequana_taxonomy --file1 Test_R1.cutadapt.fastq.gz
 --file2 Test_R2.cutadapt.fastq.gz --database <database_path>
```

Open the local HTML file taxonomy/kraken.html. An example is available in Krona example

### 2.3.5 Variant calling pipeline

The following example will show how to initialise and run the variant calling pipeline on a pair of FastQ files. For testing purposes, you can download R1 and R2 files that contain only 1500 reads. Copy them in a local directory.

Note that this does the variant calling + snpEff + coverage. See more information in the [Variant Calling](variant_calling) section.

Make sure you have installed the pipeline:

```bash
pip install sequana_variant_calling --upgrade
```

The variant calling requires input files. Since you want to map your reads onto a reference, you must have a reference. Besides, you may want to annotate your results with a specific annotation file. So, let us download those files first.

### Get the genbank reference

You can use [BioServices](https://github.com/biopython/BioServices) to download those files.

Assuming the reference is **K01711.1** (Measles virus), we first need to fetch the genbank file from NCBI:

```python
from bioservices import EUtils
eu = EUtils()
data = eu.EFetch(db="nucore", id="K01711.1", rettype="gbwithparts", retmode="text")
with open("measles.gbk", "w") as fout:
    fout.write(data.decode())
```

### Get the FASTA reference

We will also get the FASTA from ENA:

```python
from bioservices import ENA
ena = ENA()
data = ena.get_data('K01711', 'fasta')
with open("measles.fa", "w") as fout:
    fout.write(data.decode())
```

Assuming the genbank and reference have the same name, you can simply type:

```python
from sequana.snpeff import download_fasta_and_genbank
download_fasta_and_genbank("K01711", "measles")
```
Run the pipeline

```
sequana_variant_calling --input-directory . --reference measles.fa --annotation measles.gbk
cd variant_calling
sh variant_calling.sh
```

Wait and see. If the run is succesful, you can just type

```
maker clean
```

to remove some temporary files. Finally, open the file `index.html` and explore summary HTML report pages (multiqc page). Then, you can go to individual HTML report page for each sample. The individual report page are in `report_SAMPLENAME/summary.html`.

About the configuration file

We strongly recommend to look at the configuration file `config.yaml` and to check or change the parameters according to your needs. In principle, the reference and annotation file have been set up for you when initiating the pipeline.

For example, you should see those lines at the top of the config file:

```
annotation_file: measles.gbk
reference_file: measles.fa
```

**Warning:** In the configuration file, in the mark_duplicates section, some output files are huge and requires temporary directory on cluster.

**Warning:** in the configuration file (coverage section), you may need to decrease the window size for short genomes.

2.3.6 De novo

The denovo_assembly pipeline can be initialised in the same way:

```
sequana_denovo --input-directory . --working-directory denovo_test
```

Go to the `denovo_test` directory and edit the config file.

**Warning:** this is very time and computationally expensive. The **digital_normalisation** section is one that controls the memory footprint. In particular, you can check change max-tablesize to a small value for test-purposes (set the value to 3e6).
2.3.7 RNA-seq

See more information in the RNA-seq section. The following example will show you how to initialise and run the RNAseq pipeline on a couple of FastQ files (in single-end mode). The data comes from a sequencing (using HiSeq2500 technology) of a saccharomyces cerevisiae strain. For testing purposes, you can download Fastq1 and Fastq2 files that contain only 100,000 reads. Copy them in a local directory.

**Initialise the pipeline**

Call sequana standalone as follows:

```
sequana_rnaseq --working-directory EXAMPLE
```

This command download the pipeline and its configuration file. The configuration file is prefilled with adapter information and input data files found in the input directory provided. You can change the configuration afterwards.

Go to the project directory and execute the script

```
cd EXAMPLE
sh rnaseq.sh
```

**Get the fasta and GFF reference**

Assuming the reference is Saccer3 (Saccharomyces cerevisiae), we first need to fetch the fasta and the GFF files from SGD before to run the pipeline:

```
mkdir Saccer3
cd Saccer3
wget http://hgdownload.cse.ucsc.edu/goldenPath/sacCer3/bigZips/chromFa.tar.gz
tar -xvzf chromFa.tar.gz
cat *.fa > Saccer3.fa
wget http://downloads.yeastgenome.org/curation/chromosomal_feature/saccharomyces_˓
→cerevisiae.gff -O Saccer3.gff
rm -f chr*
cd ..
```

**Warning:** All files (fasta, GFF, GTF...) used in RNA-seq pipeline must have the same prefix (Saccer3 in the example) and must be placed in a new directory, named as the prefix or not.

**Warning:** For the counting step, the RNA-seq pipeline take only GFF files. GTF and SAF files will be integrated soon.
Initiate the pipeline

```
sequana_rnaseq --genome-directory Saccer3 --aligner bowtie2
```

Run the pipeline

On local:

```
snakemake -s rnaseq.rules --stats stats.txt -p -j 12 --nolock
```

On SGE cluster:

```
snakemake -s rnaseq.rules --stats stats.txt -p -j 12 --nolock --cluster-config cluster_config.json
```

On slurm cluster:

```
sbatch snakemake -s rnaseq.rules --stats stats.txt -p -j 12 --nolock --cluster-config cluster_config.json
```

2.3.8 Singularity and Sequanix

**Warning:** FOR LINUX USERS ONLY IF YOU WANT TO USE SEQUANIX. YOU CAN STILL USE THE SEQUANA STANDALONE

Here we will use a singularity container to run Sequanix and the quality pipeline to analyse local data sets stored in your /home/user/data directory.

First, Install singularity (http://singularity.lbl.gov/). Check also the Installation for information.

Second, download this specific container:

```
singularity pull --name sequana.img shub://sequana/sequana
```

This is about 1.5Go of data. Once downloaded, you can play with the container in shell or exec mode.

**shell** mode means that you enter in the container where you have an isolated environment. Because the isolated environment is protected, only the directory from where you start singularity, and optional bound directories are writable. So, if you want to read/write data in a specific directory, you must use the -B option (see section bind path here below):

```
singularity shell -B /home/user/data:/data sequana.img
```

Once in the container, you should see a prompt like this:

```
Singularity: Invoking an interactive shell within container...
Singularity sequana-sequana-release_0_5_2.img:~/Work/github/sequana/singularity>
```

Just move to the *data* directory:
You should see your input files. You can now analyse your data following the quality pipeline tutorial (top of the page), or use Sequanix:

```bash
sequanix -i . -w analysis -p quality_tutorial
```

In `exec` mode, this is even simpler:

```bash
singularity exec sequana.img sequanix
```

or with pre-filled parameters:

```bash
sequanix -i . -w analysis -p quality_tutorial
```

A Sequanix window should appear. You can now follow the Sequanix tutorial *Sequanix: GUI for snakemake workflows*.

### binding path (Mounting)

If you have data on a non standard path or want to mount a path so that the container can see it, use the binding method (see also above).

Imagine that your data on the host machine is located on `/projets/1/data` and that the file to analyse is called `virus.bed`, you can use the `sequana_coverage` tool as follows to analyse your data:

```bash
singularity exec -B /projets/1/data/:/data sequana.simg sequana_coverage --input /data/ →virus.bed
```

Here we bind the `/projets/1/data` directory (host) on the `/data` directory available in the container. Other directories available within the container are `/mounting` and `/scratch`.

### 2.4 Pipelines

Sequana ships many pipelines related to NGS. Some are very simple (fastqc, demultiplex, downsampling), others are real-life NGS pipelines used in production.

**Warning:** Since v0.8.0, pipelines are now independent from **Sequana**. They must be installed separately and their dependencies must also be installed by the user/developer.

#### 2.4.1 Quick Start

If **Sequana** is installed, installing a pipeline is straightforward. For example, to install the variant calling pipeline:

```bash
pip install sequana_variant_calling --upgrade
```

Since version 0.8.1, you can check whether you have the required dependencies. If not, an error message will appear anyway:

```bash
sequana_variant_calling --deps
```
Once those dependencies are available, you can run the pipeline:

```
sequana_variant_calling --help
```

### 2.4.2 Overview

In **Sequana** parlance, a pipeline is an application based on Snakemake that consists of a Snakefile and a configuration file. Although since v0.8.0, we augmented a pipeline with other optional files such as a schema to check the config file, a logo, a dag image representing the pipeline, a requirements file with external dependencies and so on.

All pipelines are based on Snakemake. For a tutorial, you can have a look at the Snakemake page or online-tutorials (e.g. [http://slowkow.com/notes/snakemake-tutorial/](http://slowkow.com/notes/snakemake-tutorial/)).

---

**Note:** **Pipeline naming convention**

A pipeline is named `sequana_pipelines_name` where name is to be replaced by the pipeline name. The name can contain underscores. For instance, the `variant_calling` pipeline is called `sequana_pipelines_variant_calling`. Actually, we have aliases and pipeline have usually a short name where `_pipelines` is dropped. So you can refer to a pipeline as `sequana_pipelines_variant_calling` or `sequana_variant_calling`. The reason for having the long and short versions is to avoid conflict name with Sequana standalones. For instance, the `sequana_coverage` tool exists. It is a standalone that study the coverage on a unique sample. We also have a pipeline to analyse several samples in parallel. Therefore the `sequana_pipeline_coverage` pipeline has no alias.

Future version will use the short version only.

---

### 2.4.3 Installation

Given its name, and provided you have installed Sequana, you can install the pipeline `name` using:

```
pip install sequana_name
```

where name is replaced by the pipeline name. For instance:

```
pip install sequana_fastqc
```

Since, you want to be up-to-date, add the `--upgrade` argument:

```
pip install sequana_fastqc --upgrade
```

---

### 2.4.4 Usage

Each pipeline is different. We recommend to look this complementary section **Sequana, a pipeline construction facilitator**. Generally speaking, the `--help` argument should be sufficient to run most of the pipelines:

```
sequana_name --help
```

The input arguments `--input-directory`, `--input-pattern` and `--input-readtag` will help you selecting the input data for the pipeline. Then, you will have to introspect the help and the documentation of the pipelines. Each pipeline has its own repository and living documentaion, which are available in the link here below.
2.4.5 List pipelines

This is a non-exhaustive list of pipelines

demultiplex
fastqc
Mapper
Ribofinder
Pacbio QC
quality_control
RNA-seq
Variant Calling

Please see the https://github.com/sequana organisation to get the full list.

2.5 Gallery

General-purpose examples for sequana library. The following examples illustrates how Sequana library itself can be used to read and create some plots used within the pipelines.

2.5.1 Quality histogram a la fastQC

Get a data set example

```python
from sequana import sequana_data
dataset = sequana_data("test.fastq.gz")
```

Create a FastQC instance

```python
from sequana import FastQC
qc = FastQC(dataset, verbose=False)
```

plot the histogram

```python
qc.boxplot_quality()
```
2.5.2 read length histograms pacbio data

QC pacbio example

First, let us get a data set example. Note the .bam extension

```python
from sequana import sequana_data
dataset = sequana_data("test_pacbio_subreads.bam")
```

Total running time of the script: (0 minutes 0.061 seconds)

Download Python source code: plot_fastqc_hist.py

Download IPython notebook: plot_fastqc_hist.ipynb
Sequana, Release 0.14.0

(continued from previous page)

FileNotFoundError: unknown file test_pacbio_subreads.bam. Type sequana_data() to get a...

list of valid names

Create a sequana.pacbio.BAMPacbio instance

```python
from sequana.pacbio import PacbioSubreads
qc = PacbioSubreads(dataset)
```

Traceback (most recent call last):
  File "/home/docs/checkouts/readthedocs.org/user_builds/sequana/checkouts/master/doc/...
    line 467, in execute_script
      exec(code_block, example_globals)
  File "<string>", line 2, in <module>
NameError: name 'dataset' is not defined

plot the histogram of read length

```python
qc.hist_read_length()
```

Traceback (most recent call last):
  File "/home/docs/checkouts/readthedocs.org/user_builds/sequana/checkouts/master/doc/...
    line 467, in execute_script
      exec(code_block, example_globals)
  File "<string>", line 1, in <module>
NameError: name 'qc' is not defined

plot the histogram of the SNRs for each base

```python
qc.hist_snr()
```

Traceback (most recent call last):
  File "/home/docs/checkouts/readthedocs.org/user_builds/sequana/checkouts/master/doc/...
    line 467, in execute_script
      exec(code_block, example_globals)
  File "<string>", line 1, in <module>
NameError: name 'qc' is not defined

Total running time of the script: (0 minutes 0.000 seconds)

Download Python source code: plot_qc_pacbio.py
Download IPython notebook: plot_qc_pacbio.ipynb

2.5.3 Coverage module example

```python
from sequana import GenomeCov
from sequana import sequana_data
bedfile = sequana_data("JB409847.bed")
```

Traceback (most recent call last):
  File "/home/docs/checkouts/readthedocs.org/user_builds/sequana/checkouts/master/doc/...
    line 467, in execute_script
      exec(code_block, example_globals)
  File "<string>", line 1, in <module>
NameError: name 'qc' is not defined

(continues on next page)
Reading input BED file

```python
gc = GenomeCov(bedfile)
```

Traceback (most recent call last):
  File "/home/docs/checkouts/readthedocs.org/user_builds/sequana/checkouts/master/doc/sphinxext/sphinx_gallery/gen_rst.py", line 467, in execute_script
    exec(code_block, example_globals)
  File "<string>", line 1, in <module>
NameError: name 'bedfile' is not defined

Select a chromosome (first and only one in that example)

```python
chrom = gc[0]
print(chrom)
```

Traceback (most recent call last):
  File "/home/docs/checkouts/readthedocs.org/user_builds/sequana/checkouts/master/doc/sphinxext/sphinx_gallery/gen_rst.py", line 467, in execute_script
    exec(code_block, example_globals)
  File "<string>", line 1, in <module>
NameError: name 'gc' is not defined

Compute running median and z-score telling the algorithm that the chromosome is circular.

```python
chrom.running_median(n=5001, circular=True)
chrom.compute_zscore()
print(chrom.get_centralness())
```

Traceback (most recent call last):
  File "/home/docs/checkouts/readthedocs.org/user_builds/sequana/checkouts/master/doc/sphinxext/sphinx_gallery/gen_rst.py", line 467, in execute_script
    exec(code_block, example_globals)
  File "<string>", line 1, in <module>
NameError: name 'chrom' is not defined

Plotting

```python
chrom.plot_coverage()
```

Traceback (most recent call last):
  File "/home/docs/checkouts/readthedocs.org/user_builds/sequana/checkouts/master/doc/sphinxext/sphinx_gallery/gen_rst.py", line 467, in execute_script
    exec(code_block, example_globals)
  File "<string>", line 1, in <module>
NameError: name 'chrom' is not defined
2.5.4 BAM module example

Plot histogram of MAPQ values contained in a BAM file

first import the relevant modules

```python
from sequana import BAM, sequana_data
```

Get a data set (BAM file) for testing

```python
from sequana import BAM, sequana_data
datatest = sequana_data('test.bam', 'doc')
```

Use `sequana.bamtools.BAM` class to plot the MAPQ histogram

```python
b = BAM(datatest)
b.plot_bar_mapq()
```

# for some unknown reasons, we must call show in this example for this image to appear in the doc
```python
from pylab import show
show()
```

2.5. Gallery
2.5.5 Kraken module example

In Sequana, we provide tools to quickly assess the taxonomic content of reads (FastQ). It is based on the Kraken and Krona software. Sequana bridges the gap between those tools, Kraken databases and a simple interface to get a quick taxonomic overview of the FastQ content.

For more information, please see the sequana.kraken module documentation. Note that this feature is also part of the quality_control pipeline.

Although we will use the Sequana library hereafter, note that there is also a standalone application named sequana_taxonomy.

Context

Running the kraken analysis takes some time so we do provide an output directly. The output file can be analysed within Sequana to plot a Pie chart but also Javascript Krona results. The kraken format is as follows:

```
C     HISEQ:426:C5T65ACXX:5:2301:5633:7203 11234 203 0:2 11234:1 0:1 11234:1 0:2
     →11234:1 0:13 11234:1 0:1 11234:1 0:3 11234:1 0:16 11234:1 0:5 11234:1 0:6 11234:1 0:13
     →A:31 0:33 11234:1 0:29 11234:1 0:7
C     HISEQ:426:C5T65ACXX:5:2301:5815:7120 11234 203 0:4 11234:1 0:12 11234:1 0:22
     →11234:1 0:1 0 11234:1 0:5 11234:1 0:7 11234:1 0:5 A:31 0:3 11234:1 0:22 11234:1 0:18
     →11234:1 0:24 11234:1
```

Each row correspond to a read in the FastQ file(s). The first column is either C (classified) or U (unclassified) and the third column contains the taxon the most relevant.

The taxon are not readable so we first need to get the scientific names. Besides, the lineage would be useful. This is done in Sequana using the sequana.kraken.KrakenResults. See following example.

Example

In the following example, we use the results of a kraken analysis. The original toy data files contains 1500 reads mostly related to Measles virus

```python
from sequana import KrakenResults, sequana_data
test_file = sequana_data("test_kraken.out", "examples")
k = KrakenResults(test_file)
df = k.plot(kind='pie', delete_krona_file=True)
print(df)
```
Out:

```
name: Measles virus genotype G3                  1.010101
name: Measles virus genotypes and isolates     3.030303
name: Measles virus strain Edmonston-Zagreb     8.080808
name: Measles virus strain MVi/California.USA/8.04 8.080808
Unclassified                                   10.101010
name: Measles virus                               69.696970
dtype: float64
```

Note that only a subset of taxons are shown in the pie chart that is those that cover at least 1% of the total reads. Others are put together and labelled "others"

A more interactive plot can be obtained using Krona if installed:

```
from sequana import KrakenResults, sequana_data
test_file = sequana_data("test_kraken.out", "examples")
import easydev
if easydev.cmd_exists("ktImportText"):
    k = KrakenResults(test_file)
    k.to_js()  # The output file name is krona.html by default
```

An example is available in Krona example
2.5.6 Pipeline statistics

First, let us get the data

```python
from sequana_pipetools.snakertools import get_pipeline_statistics
df = get_pipeline_statistics()
```

Plot number of rules per pipeline

Note that pacbio_qc is self-content

```python
from pylab import title, tight_layout
df.sum().plot(kind="barh")
title("Number of rules per pipeline")
tight_layout()
```

Proportions of rules re-used

Amongst the rules, about a third of the rules are not used at all in the pipelines. There are two reasons: either they were part of previous pipeline versions and were discarded in favour of new tools, or there were used for testing and kept in
Then, we can see that a third of the rules are used only once. And finally, about a third used more than once.

```python
from pylab import clf, pie
from collections import Counter

count = Counter(df.sum(axis=1))
values = list(count.values())
times = list(count.keys())
clf()
pie(list(count.values()), labels=["{} used {} times".format(x,y) for x,y in zip(values,times)])
```

Total running time of the script: (0 minutes 0.162 seconds)

Download Python source code: plot_pipeline_stats.py
Download IPython notebook: plot_pipeline_stats.ipynb
2.5.7 Running median example

Plot running median on a data set

```python
from sequana.running_median import RunningMedian
from pylab import *

N = 1000
X = linspace(0, N-1, N)

# Create some interesting data with SNP and longer over # represented section.
data = 20 + randn(N) + sin(X*2*pi/1000.*5)
data[300:350] += 10
data[500:505] += 100
data[700] = 1000

plot(X, data, "k", label="data")
rm = RunningMedian(data, 101)
plot(X, rm.run(), 'r', label="median W=201")

from sequana.stats import moving_average as ma
plot(X[100:-100], ma(data, 201), 'g', label="mean W=201")
grid()
```

(continues on next page)
2.6 Case Examples

2.6.1 Effect of the trimming on SNPs detection

**Description** Effect of trimming (or not trimming) on the SNPs detection.

In this case example, we will take a paired-end dataset, and apply the quality pipeline using trimming quality (removing bases with quality below 30). Then, we will run the variant calling pipeline to perform the mapping on a reference and detect SNPs.

We will repeat this analysis without trimming low quality reads at all.

We will finally compare the two sets of SNPs showing that the trimming quality is not important in this example. Meaning that the mapping tool used (freebayes) is able to cope with low quality reads.

**The data**

We will use a paired-end dataset (MiSeq 250bp). It contains 250,000 reads (X2). The organism sequenced is *Bordetella*. As a reference, we use the ENA accession CP010347.1. The data will be posted later but the original data were generated at Pole Biomics (Institut Pasteur) and named Tohama-R0_S4_L001_R1_001 from which we used only the first 250,000 reads.

Here is a boxplot of the base quality across the reads showing that the quality is quite high and falls below 30 after 200 bases.
Quality pipeline

Assuming DATA (fastq.gz files) are in <DIR1> directory, type this command to create the quality pipeline and config file automatically:

```
sequana_quality_control --input-directory <DIR1> --working-directory trimming
```

Then go to the project and execute the pipeline:

```
cd trimming
snakemake -s quality_control.rules -p -j 4 --forceall
```

Note: you can also use Sequanix to help in the configuration design.

The final cleaned reads are in trimming/Tohama-R0_S4_L001/report/outputs (refered to <DIR2> hereafter) and named after the project: (trimming_R1_.cutadapt.fastq.gz and trimming_R2_.cutadapt.fastq.gz). These two files should be used later as the input of the variant_calling pipeline, as shown hereafter.

There is no adapters in the data so in the config file, the adapter sections are empty (no forward or reverse adapters). Note, however, that bad quality bases below 30 (default) are removed. In order to set the quality to another values, use sequana with the --quality option

See also:

See the Tutorial and User guide and reference sections for more details.
Quality pipeline (No trimming)

Repeat the previous two steps. In the first step, change the adapter section (cutadapt) to set the quality to zero (this prevents the trimming of bad quality bases):

```bash
cutadapt:
  quality: 0,0
```

Change the project name e.g. `no_trimming` as a tag to the project in the first step and `variant_no_trimming`.

Variant analysis

The output of the quality pipeline will be the input of the variant calling pipeline:

```bash
sequana_variant_calling --input-directory <DIR2> --working-directory variant_trimming
```

Here you need to make sure that the `config.yaml` configuration file has the correct reference. See the Tutorial section (variant section).

```python
reference = "CP010347"
from bioservices import EUtils
eu = EUtils()
data = eu.EFetch(db="nuccore",id=reference, rettype="gbwithparts", retmode="text")
with open("data.gbк", "w") as fout:
    fout.write(data.decode())
from bioservices import ENA
ena = ENA()
data = ena.get_data(reference, 'fasta')
with open("data.fa", "w") as fout:
    fout.write(data.decode())
from sequana import snpeff
v = snpeff.SnpEff("data.gbк")
```

Edit the config.yaml to change those sections:

```yaml
# snpEff parameter
snpeff:
  do: yes
  reference: "data.gbк"

# Bwa parameter for reference mapping
bwa_ref:
  reference: "data.fa"
```

Run the analysis:

```bash
cd variant_trimming
snakemake -s variant_calling.rules --stats report/stats.txt -p -j 4 --forceall
```

Once done, you should have VCF files in `variant/report/` named `cutadapt.ann.vcf`
SNPs results comparison

You should now have two VCF files. Here below we plot the read depth versus strand balance. The color will indicates the overall freebayes score (normalised by the largest score). A good candidate should have large score and balance value around 0.5. The y-axis shows the read depth.

```python
from pylab import *
from sequana import freebayes_vcf_filter

vcf1 = freebayes_vcf_filter.VCF("variant/report/cutadapt.ann.vcf")
vcf2 = freebayes_vcf_filter.VCF("variant_no_trimming/report/variant_no_trimming.ann.vcf")

df1 = vcf1.filter_vcf()._vcf_to_df()
df2 = vcf2.filter_vcf()._vcf_to_df()

subplot(1,2,1)
scatter(list(df1.strand_balance.values), list(df1.depth.values), c=list(df1.freebayes_score.values/1240))
xlabel("strand balance")
ylabel("Depth")
grid()
ylim([0,90])

subplot(1,2,2)
scatter(list(df2.strand_balance.values), list(df2.depth.values), c=list(df2.freebayes_score.values/1240))
grid()
ylim([0,90])
title("Trimming quality (left) vs no trimming (right)")
```
In this figure the LHS (trimming) 294 SNPs were found while in the RHS (no trimming) 309 were found. The additional SNPs all have low coverage below 20. A third of them have low balance strand.

There is one SNP found in the trim case not found in no_trim. However, it is marginal with strand balance of 0.12, depth of 11, frequency of 0.73 and one of the lowest score.

**Conclusions**

The detection of SNPs does not suffer from not trimming low quality bases. Actually, some new SNPs are found. However, the are usually not significant (low depth, low score or unbalanced). Interestingly, the distribution of the SNPs in the depth vs strand balance plane seems to be more centered on strand balance=0.5. We also notice that the depth is 10% better which means that the low quality bases have contributed to the improvements of the depth and freebayes scores. It could be interesting to extend the analysis to more data, lower quality, or higher quality threshold. Note also that because there are more low quality bases, there much more false alarms; However setting a freebayes score threshold around 5 removes most of them.

### 2.7 Applications (standalone)

**Contents**

- Applications (standalone)
  - Sequanix: GUI for snakemake workflows
  - sequana_coverage
  - sequana_summary
### 2.7.1 Sequanix: GUI for snakemake workflows

**Overview**  
A Graphical User Interface (GUI) for Sequana pipelines and any Snakemake-based workflows.

**Status**  
Production

**Name**  
sequanix

This GUI can be used to load Snakefile and their configuration file. A working directory has to be set. Once done, the configuration file can be changed in the GUI. Finally, one can run the snakefile and see the progress. Tooltips are automatically created from the configuration file (if documented).

Since snakemake has the ability to run jobs locally or on a cluster, this application can also be run either locally or a distributed computing platform (e.g., cluster with slurm scheduler). Of course, this means you can use a X environment on your cluster (ssh -X should do it).

Just type **sequanix** in a shell.

**Note:** tested under Linux only. However, Mac and Windows users should be able to use it since it is based on Python and PyQt. Again, we strongly advice to use Anaconda to install all required dependencies

Here is a snapshot.
See also:

see Sequanix Tutorial for details
2.7.2 sequana_coverage

Description  Show coverage and interval of confidence to identify under and over represented genomic regions.

Status  Production

Help  please use sequana_coverage --help

Sequana  See GenomeCov to use the coverage in your own script.

Gallery  See examples in the gallery

Starting from a BED file and its reference, one can use this command in a shell:

```
sequana_coverage --input JB409847.sorted.bed -o --reference JB409847.fa --show-html
```

It creates an HTML report with various images showing the coverage and GC versus coverage plots. It also provides a set of CSV files with low or high coverage regions (as compared to the average coverage).

See also:

the underlying algorithm is described in details in the documentation (sequana.bedtools.GenomeCov).

2.7.3 sequana_summary

Description  Prints basic statistics about a set of NGS input files. Currently handles Fastq (gzipped or not) or BED files (coverage).

Usage  sequana summary file1.fastq.gz

2.7.4 sequana_mapping

Description  a simple application to map reads onto a genome given one or two FastQ files (gzipped) and a reference.

```
sequana_mapping --file1 H1_R1.fastq.gz --file2 H1_R2.fastq.gz --reference temp.fa
```

will map all reads on the reference using bwa.

2.7.5 sequana_taxonomy

Description  Creates a HTML document with Krona and pie chart of taxonomic content of a FastQ file (paired or not). Uses Kraken, Krona and a dedicated Sequana database.

Help  sequana_taxonomy --help

Status  Production

Sequana  see sequana.kraken

Gallery  see Kraken module example

You will need to download databases. We provide a toy example:

```
sequana_taxonomy --download toydb
```

and the official kraken DB (4Gb):
sequana_taxonomy --download minikraken

A database of 8Gb is available. See https://github.com/sequana/data/tree/master/sequana_db1 for instructions and details (bacteria, viruses, human, organelles, ...).

### 2.7.6 fastq related

**Description** count number of reads and lines

`sequana fastq test.fastq.gz --count-reads`

**Description** extracts head of fastq files

`sequana fastq test.fastq.gz --head 10000 -o output.fastq.gz`

### 2.7.7 sequana_compressor

**Description** standalone on top of the compressor pipeline to compress/decompress FastQ files in different formats, recursively and using multithreaded and multicore tools.

**Status** Production

### 2.8 Sequanix Tutorial

- Quick Installation
- Introduction
  - Sequana pipeline: the quality control example
    - Prerequisites: get some data
    - Select the quality control pipeline
    - Select the input data (directory or files)
    - Select the working directory
    - Fine tune the config parameters
    - Save, check and run the project
    - Stopping a running analysis:
      - Start Sequanix with pre-defined values
  - Generic pipeline: a minimalist example with no configuration file
    - Prerequisites: get some FastQ files
    - The analysis
    - Save, check and run the project
    - Start Sequanix with pre-defined values
  - Generic pipeline: a minimalist example with a configuration file
• Dialogs and running analysis locally or on a cluster
  – The Sequanix browser and the preferences dialog
    * sequana_fox, the sequanix home-made browser
    * Preferences dialog
  – From a local to cluster analysis
    * Running analysis locally
    * Running analysis on a cluster
    * Other Snakemake options
• FAQs
  – How to run Sequanix on a SLURM cluster.
  – What to do if a RUN fails

2.8.1 Quick Installation

Sequanix is shipped with Sequana. If you use conda, just type:

```
conda install sequana
```

Please see Installation for details. The standalone name is sequanix.

2.8.2 Introduction

Sequanix is a graphical user interface (GUI) that can be used to run Snakemake workflows. The standalone name is sequanix (small caps) and is part of Sequana library.

The primary goal was to provide a GUI to easily run Sequana pipelines (designed as Snakemake workflows).

However, we extended the interface so that it can handle other Snakemake workflows, referred to as Generic pipelines in the GUI.

A tutorial in form of a video is available in another repository: video/tutorial

In this section, we first show how to run one of our Sequana pipeline (quality control pipeline). Second, we show how to run Generic pipelines that are not part of Sequana. For these two examples, the computation is done locally. However, one strength of Snakemake pipelines is that they can be executed on various cluster without changing the pipeline itself. This is also possible via Sequanix as explained in the Running analysis on a cluster section (SLURM and SGE frameworks).

Snakemake pipelines are made of 2 parts: a pipeline and an optional configuration file; The pipeline may be called Snakefile. It contains the code of the pipeline itself. Keep in mind that in the Snakefile, developer may link the pipeline to an external configuration file: the config file, which is encoded in YAML or JSON format.
Fig. 1: Snapshot of the Sequanix graphical user interface (GUI)
2.8.3 Sequana pipeline: the quality control example

Prerequisites: get some data

The following example will show how to run a quality control pipeline on a pair of FastQ files. The data comes from a sequencing platform (using HiSeq technology) of a Measles virus. For testing purposes, you can download R1 and R2 files that contain only 1500 reads. Copy the two files in a local directory (let us call it testing) and start sequanix.

```
cd testing
sequanix
```

Select the quality control pipeline

First you need to select the pipeline of interest (here the `quality_control`). In the following figure, you need to

1. select the `sequana pipelines` tab (arrow 1),
2. select the `pipeline section` tab (arrow 2)
3. select the pipeline in the dropdown box (arrow 3)

Once done, the configuration file of the pipeline will be loaded in the `Config parameters` tab (arrow 4).

If a file named `config_cluster.json` is defined by the developer in the pipeline module, then it is loaded automatically in the snakemake dialog and used if the pipeline is executed on a cluster.

Select the input data (directory or files)

Once the pipeline is selected, you need to give information about the location of the input data. Generally, the pipelines take fastq.gz files as input.

In general we have many samples, so you need to select the `Input directory` tab. If you have only one or two files, you may use the other tab (`Input sample(s)`).

Here, we consider the first case only (directory). First click on the red Browse button (figure below) to select the directory where is stored the data.

By default, we assume that there is a special tag in the filenames (_R1_ or _R2_) but one can change it to another pattern. Note also that we expect by default the input files to end up in fastq.gz.

So by default if you select a directory, all files ending in fastq.gz will be selected.

- **Note:** if FastQ files are stored in various directories, use the `Optional pattern`. For instance if samples are stored in sub directories, use `*/fastq.gz`

- **Note:** The directory browser can create directories
Select the working directory

As above, when clicking on the browser button, you may select an existing one or create a new one.

Fine tune the config parameters

**Warning:** Sequana pipelines may be complex with several dependencies on external tools. We would recommend users to look at the online documentation for help (e.g., Tutorial, Pipelines).

One major interest of Sequanix is that the Snakemake configuration file is loaded and can then be changed dynamically. In other word, you do not need to use an esoteric text editor, which may be the only option on a cluster.

Moreover, the loaded configuration file has other advantages:

- file can be selecting thanks to a file browser. If no file is selected, the button is red (green otherwise)
- Some buttons have dedicated widgets (e.g. in the figure above, the number of threads has its own dropbox limited typing errors)
- Boolean fields have their own checked buttons
- etc

**Note:** For developers: please see the Documenting the configuration file section to see how to write your configuration to have the widgets loaded automatically.
Save, check and run the project

Once the parameters have been set, it is time to save the project. You can either click the yellow box Save in the bottom bar or the Ctrl+S shortcut.

The configuration and pipelines files are then save in the working directory defined above. If the files already exists, a dialog box ask you to confirm that you want to overwrite the existing files.

You can then check the pipeline by clicking the Show Pipeline button or use Ctrl+D shortcut. For simple pipeline, this may not be very useful but for complex dynamic pipelines where parts may be switched off, this may be convenient.

![Fig. 3: A dialog showing the DAG (directed acyclic graph) with dependencies in the analysis pipeline](image)

Finally, once saved, the Run button should be clickable. Click on it or use Ctrl+R shortcut. The output of Snakemake will be shown and the progress bar will move showing the stage of the analysis.

**Warning:** with long analysis, the progress bar may be stalled for a while. It may even stay at 0% for a long time. Just be patient.

**Stopping a running analysis:**

If you realise that you made a mistake in the configuration or simply want to stop the current analysis, click the Stop button
Start Sequanix with pre-defined values

If you use Sequanix regularly, it may be convenient to start the standalone with pre-filled values. For instance, to pre-fill the input directory, the working directory and the pipeline itself start Sequanix as follows:

```
sequanix -i . -p quality_control -w analysis
```

For help, please type:

```
sequanix --help
```

2.8.4 Generic pipeline: a minimalist example with no configuration file

In this section we will use a very simple Snakefile that reads FastQ files (gzipped) and counts the number of reads (not lines). The results are then summarised into a file named `summary.txt`. For those who are curious, here is the Snakefile.

```python
import glob
import os

# Input parameter to be changed
directory = "./"

directory = os.path.abspath(directory) + "/
filenames = glob.glob(directory + "*.fastq.gz")
samples = [os.path.split(this)[1].replace(".fastq.gz", "")
  for this in filenames]

rule all:
  input: "summary.txt"

rule summary:
  input: expand("count_{sample}.txt", sample=samples)
  output: "summary.txt"
  shell: "cat count_*_.txt > summary.txt"

rule count:
  input: directory + "{sample}.fastq.gz"
  output: temp("count_{sample}.txt")
  shell: "gunzip -c {input} | wc - | awk '{print $1/4}'> {output}"
```

Note:

- In this example, the directory where to find the data is hardcoded so you must change it (see highlighted line in the code below).
- This example does not depend on any external configuration file. We will see later on to combine this Snakefile with a configuration file where the directory can be set.
Prerequisites: get some FastQ files

To run the pipeline, we first need to get some FastQ files (zipped). We need to store them in a data directory. Create one and move into the directory as follows:

```
mkdir data
cd data
```

If you do not have FastQ files, get the following ones:

- R1
- R2

You will also need the Snakefile (pipeline) itself:

- minimalist

**Warning:** if the data and pipeline are in a different directories, you need to change the highlighted line (line 5) to set the **directory** name specifically.

Once ready, start **Sequanix** in a shell:

```
sequanix
```

The analysis

Similarly to the Sequana pipeline case, you need to select the pipeline as follows:

1. Select the *Generic pipelines* tab (arrow 1)
2. Select the *pipeline section* tab (arrow 2)
3. Click on the browse button to select the pipeline file (minimalist.rules)

There is no configuration file so we can now save and run the project:

**Save, check and run the project**

Same as for a Sequana pipeline

**Start Sequanix with pre-defined values**

Alternative way to start sequanix with pre-filled values for the working directory and the pipeline file:

```
sequanix -w analysis -s minimalist.rules
```
2.8.5 Generic pipeline: a minimalist example with a configuration file

In this section, we use a pipeline that is almost identical to the previous one.

```python
import glob
import os

configfile = "minimalist.yaml"
directory = config['data_directory']
directory = os.path.abspath(directory) + "/
filenames = glob.glob(directory + "*.fastq.gz")
samples = [os.path.split(this)[1].replace(".fastq.gz", "")
    for this in filenames]

rule all:
    input: "summary.txt"

rule summary:
    input: expand("count_{sample}.txt", sample=samples)
    output: "summary.txt"
    shell: "cat count_*txt > summary.txt"

rule count:
    input: directory + "{sample}.fastq.gz"
    output: temp("count_{sample}.txt")
    shell: "gunzip -c {input} | wc - | awk '{{print $1/4}}' > {output}"
```

The only difference is on line 4 and 5: the previously hard-coded variable directory is now extracted from an external configuration file called minimalist.yaml.

Here are the links to get the Snakefile and the configuration file.

- minimalist file with configuration
- configuration

Similarly to the previous example you would need some FastQ files (see Prerequisites: get some FastQ files). Once done, start sequanix. Here you would need to load the pipeline and set the working directory but also to load the config file. When you load the config file, you should see something equivalent to the following figure: the configuration file is shown in the Config parameters section:

You can see here that the configuration file (a single parameter data_directory) is interpreted and a widget is available to select the directory where to find the data (for developers, please see Developer guide section).

The rest of the analysis works as above.
Sequana, Release 0.14.0

Chapter 2. User guide and reference

![Sequana GUI screenshot]

**Pipeline control**

- **Snakefile**: Browse to `../quana/doc/minimalist.yaml`

**Parameters in no sections/rules**

- **data_directory**: Browse to `../Work/github/quana/doc`

**Additional Information**

Is it a local or cluster run? (Local/Cluster)
### 2.8.6 Dialogs and running analysis locally or on a cluster

So far we have used Sequanix with the default parameters.

#### The Sequanix browser and the preferences dialog

**sequana_fox, the sequanix home-made browser**

Once an analysis is finished, Sequana pipeline generally creates an HTML report. This is the reason why we added an **Open Report** button in the bottom. This opens a file browser where users can select an HTML file. The browser used by default is a home-made browser so that it can run on a cluster where no standard browser are installed.

The home-made browser, which can be used as a standalone (**sequana_fox**) is simple but should be enough for most HTML pages. There is a forward/backward capability, support for Javascript, ability to change the URL but that is pretty much all. This is mainly used to check that HTML files have been created.

#### Preferences dialog

In order to open the preferences dialog, type **Ctrl+P** or go to the **Option** menu at the top and select **Preferences**. The Preferences dialog looks like the following figure:

![Preferences dialog](image)

Fig. 4: Preferences dialog. This dialog is accessible via the menu or the short Ctrl+P. It contains general options to tune Sequanix’s behaviour.

**Brief description of the options:**

- **overwrite files** if checked, when saving a project, the existing configuration and pipelines are overwritten
- **select the browser to be used** By default the home-made browser (pyqt5) is used but one can select firefox, safari, chrome instead.
- **logging verbosity** there are 5 level of verbosity. By default, we use INFO. It may be useful to set the option to DEBUG if there are errors and you wish to provide a complete bug report to Sequana developers.
- **HTML page to open as a report** If you set a filename here, then when pressing **Open report**, instead of opening a file browser, sequanix open the file provided.
Form browser keywords  In the config parameters, if you wish to associate a parameter name with a browser widget, add the names here (separated by comma)

From a local to cluster analysis

One strength of Snakemake (and Sequanix) is that pipelines can be run locally but also on clusters using various scheduler frameworks without changing the pipeline code.

In Sequanix, we can switch between a local run or a cluster run by switching a button in the main window as shown in the figure below:

The Snakemake dialog contains 3 sub tab: the local, cluster and general tabs.

Running analysis locally

If you run the analysis locally, you do not need to change much. The only option to tune is the number of cores to be used locally. This happens in the Local tab. By default the cores parameter is set to the number of cores found on the computer. You may reduce this number if you wish.

Running analysis on a cluster

If you run the analysis on a cluster, this is a bit more complicated.

First, similarly to the local run, you may provide the number of cores to be used. This happens in the Cluster tab. Here, you can set the parameters jobs to the required number of CPUS. If you know that at a given time, you may have N jobs running, set this parameter to N. For instance, if you have 48 samples, and you perform 48 independent analysis, set jobs to 48.

Second, you must set the cluster commands. We will not provide an exhaustive documentation on this aspect, which is technical and pipeline and cluster dependent.

We provide two examples. First, let us assume the case where:

• you are on a cluster with a SLURM framework
• your jobs require less than 4 Gb of memory

Then, you must add this line in the cluster field

```
sbatch --mem=4000
```

Second, let us assume the case where:

• you are on a cluster with a SGE framework
• your jobs require 4 threads each

Then, you must add this line in the cluster field

```
qsub --pe threaded 4
```
Fig. 5: If you are on a cluster, you should switch the local mode to cluster AND you have to provide the cluster commands in the Snakemake dialog (see above).
Fig. 6: The **local** tab contains only one option to set the number of local cores to be used. By default it is the number of available cores on the machine used.
Fig. 7: The tab **cluster** contains parameters related to the execution of the Snakemake pipeline can be set (e.g. specific job scheduler information or number of CPUs to be used).
Other Snakemake options

Snakemake itself has lots of options. In the snakemake dialog, in the General tab, one can set them.

![Snakemake options](image)

Fig. 8: In the General tab, checkboxes related to Snakemake are available. Any other options can be set in the editable line at the bottom.

Here is a brief description:

- **quiet**  Do not output any progress or rule information
- **forceall**  Force the execution of the selected (or the first) rule and all rules it is dependent on regardless of already created output.
- **keep-going**  Go on with independent jobs if a job fails.
- **nohooks**  Do not invoke onstart, onsuccess or onerror hooks after execution.
- **restart-times**  Number of times to restart failing jobs (defaults to 0).
- **verbose**  Print debugging output
- **summary**  Print a summary of all files created by the workflow. The has the following columns: filename, modification time, rule version, status, plan. Thereby rule version contains the version the file was created with (see the version keyword of rules), and status denotes whether the file is missing, its input files are newer or if version or implementation of the rule changed since file creation. Finally the last column denotes whether the file will be updated or created during the next workflow execution.
- **any other options**
2.8.7 FAQS

How to run Sequanix on a SLURM cluster.

You have to connect with ssh and the -X option:

```bash
ssh -X your.cluster.address
```

Once connected, on a slurm system type:

```bash
srun --x11 sequanix
```

What to do if a RUN fails

An analysis may fail for various reasons. The errors have several origins. Most commons ones are:

- configuration file not filled properly (e.g. missing input file)
- Input data not found (e.g., bad pattern)
- bug in the pipeline (Fill an issue on http://github.com/sequana/sequana/issues)
- bug in Sequanix (fill an issue)
- cluster issue: a job is killed because not enough memory was allocated (adapt the cluster option by increasing relevant resources such as memory requirements)

By experience, the first 2 type of errors are the most common.

2.9 Developer guide

This section is a tutorial for developers who wish to improve Sequana, in particular how to include a new rule or a new pipeline.

Since v0.8.0, creating a pipeline is very easy since we provide a template.

Yet, before creating a pipeline, we will need the bricks to build it. In Snakemake terminology, a brick is called a rule.

We will create a very simple pipeline that counts the number of reads in a bunch of FastQ files. First, we will need to create the rule that counts the reads and then the pipeline. Once the pipeline is created, we will create the documentation, test and HTML reports. Finally, when you have a pipeline that creates a reports and summary file, you may want to also include a multiqc summary. We will also show how to integrate this feature inside our framework.

The rule simply counts the number of reads in a fastq file. The pipeline will only contains that unique rule.

In the remaining sections, we will explain our choice concerning the continuous integration (section Testing with pytest ) and how to add sanity check that the new code do not introduce bugs. In the Module reports section, we explain how to create new component in the HTML module reports.
2.9.1 How to write a new pipeline in Sequana (the new way)

First, you need to find a name for your pipeline. Check out the https://github.com/sequana organization page to check whether it is not yet taken. Note also that pipeline names may need to be different from all rules available in sequana.

Find a valid name

All rules and pipelines must have a unique name in Sequana. We can quickly check that a name is not already taken as follows:

```python
>>> from sequana_pipetools.snaketools import modules
>>> "count" in modules.keys()
False
```

So, let us name it `count`

Create a Snakefile

A possible code that implements the `count` rule is the following Snakefile:

```python
from sequana import sequana_data
filename = sequana_data("Hm2_GTGAAA_L005_R1_001.fastq.gz", "data")

rule count:
    input: filename
    output: "count.txt"
    run:
        from sequana import FastQ
        def count(fastq):
            return len(FastQ(fastq))
        results = dict([(filename, count(filename)) for filename in input])
        with open(output[0], "w") as fout:
            fout.write("%s" % results)
```

This is not a tutorial on Snakemake but let us quickly explain this Snakefile. The first two lines use Sequana library to provide the `filename` as a test file.

Then, the rule itself is defined on line 4 where we define the rule named: `count`. We then provide on line 5 and 6 the expected input and output filenames. On line 7 onwards, we define the actual function that counts the number of reads and save the results in a TXT file.

You can now execute the Snakefile just to check that this rule works as expected:

```
snakemake -s Snakefile -f
```

You can check that the file `count.txt` exists.

**Note:** The option `-f` forces snakemake to run the rules (even though it was already computed earlier).
A sequana pipeline

Somehow, the code above is enough. This is a valid pipeline, which is functional. Yet, we have to handle many different pipelines within our framework. Therefore, we impose some rules so that arguments are similar, testing, documentation are coherent.

This is achieved easily using the standalone sequana_start_pipeline as follows:

```
sequana_start_pipeline --name count
```

Press enter 4 times and you get your new pipeline structure that looks like:

```
|-- doc
 |  |-- conf.py
 |  |-- index.rst
 |  |-- Makefile
|-- LICENSE
|-- README.rst
|-- requirements.txt
|-- sequana_pipelines
 |  |-- count
 |   |-- config.yaml
 |   |-- count.rules
 |   |-- data
 |   |   |-- __init__.py
 |   |   |-- __init__.py
 |   |-- main.py
 |   |-- requirements.txt
 |   |-- schema.yaml
|-- setup.cfg
|-- setup.py
|-- singularity
 |  |-- Makefile
 |  |-- Singularity
|-- test
 |  |-- __init__.py
```

This is a valid Python package. What you need to do now is copy your Snakfile into ./sequana_pipelines/count/count.rules and adapt the main script sequana_pipelines/count/main.py to your needs.

Once ready, install the package:

```
python setup.py install
```

Check the documentation in the README.rst, add a test in ./test/test_main.py and you are ready to upload your package on pypi.

Ideally, you will now add a repository in https://github.com/sequana/ and add/commit/push your code.

The count rules is now part of the library, which can be checked using the same code as before:

```
>>> import sequana
>>> "count" in sequana.modules.keys()
True
```
2.9.2 Convention to design a rule

Use variables

Consider this Snakefile:

```bash
rule bedtools_genomecov:
    input:
        __bedtools_genomecov__input
    output:
        __bedtools_genomecov__output
    params:
        options = config["bedtools"]["options"]
    shell:
        bedtools genomecov {params.options} -ibam {input} > {output}
```

We tend to not hard-code any filename. So the input and output are actually variables. The variable names being the name of the rule with leading and trailing doubled underscores followed by the string `input` or `output`.

**Note:** The big advantage of designing rules with variables only: rules can be re-used in any pipelines without changing the rule itself; only pipelines will be different.

Use a config file

We encourage developers to NOT set any parameter in the `params` section of the Snakefile. Instead, put all parameters required inside the `config.yaml` file. Since each rule has a unique name, we simply add a section with the rule name. For instance:

```yaml
bedtools_genomecov:
    options: ''
```

This is a YAML formatted file. Note that there is no information here. However, one may provide any parameters understood by the rule (here `bedtools genomecov` application) in the `options` field.

We encourage developers to put as few parameters as possible inside the config. First to not confuse users and second because software changes with time. Hard coded parameter may break the pipeline. However having the `options` field allows users to use any parameters.

**See also:**

Sequana contains many pipelines that can be used as examples. See github repo

**Note:** Boolean are very permissive. One can use: true|True|TRUE|false|False|FALSE yes|Yes|YES|no|No|NO on|On|ON|off|Off|OFF
Add documentation in the rule

In sequana, we provide a sphinx extension to include the inline documentation of a rule:

```
.. snakemakerule:: rule_name
```

This searches for the rule docstring, and includes it in your documentation. The docstring should be uniformised across all rules and pipelines. Here is our current convention:

```
rule cutadapt:
    """Cutadapt (adapter removal)

    Some details about the tool on what is does is more than welcome.

    Required input:
    - __cutadapt__input_fastq

    Required output:
    - __cutadapt__output

    Required parameters:
    - __cutadapt__fwd: forward adapters as a file, or string
    - __cutadapt__rev: reverse adapters as a file, or string

    Required configuration:
    .. code-block:: yaml

        cutadapt:
            fwd: "%(adapter_fwd)s"
            rev: "%(adapter_rev)s"

    References:
    a url link here or a link to a publication.
    """

input:
    fastq = __cutadapt__input_fastq
output:
    fastq = __cutadapt__output
params:
    fwd = config['cutadapt']"fwd",
    rev = config['cutadapt']"rev"
run:
    cmd = "cutadapt -o {output.fastq[0]} -p {output.fastq[1]}" 
    cmd += " -g %s -G %s" % (params.fwd, params.rev)
    cmd += "{input.fastq[0]} {input.fastq[1]}"
shell(cmd)
```

Here is the rendering:
2.9.3 More information about pipeline design

The Snakefile/pipeline

The first thing to notice as compared to a standard Snakefile is that we use rules from Sequana only (for the moment). There are already many rules and they can be added as follows:

```python
from sequana import snaketools as sm
include: sm.module['rulegraph']
```

This will take care of finding the exact location of the module.

Second, all configuration file are named `config.yaml`.

So, your pipeline should look like:

```python
import sequana
from sequana import snaketools as sm
#sm.init("counter", globals()) # see later for explanation

cfgfile: "config.yaml"

# include all relevant rules
include: sm.modules['count'] # if included in sequana/rules

# must be defined as the final rule
rule pipeline_count:
    input:
        "count.txt"
```

The pipeline README file

In the same directory as your pipeline Snakefile, add a README.rst file. Here is a template to be used to create the documentation (replace NAME by the pipeline name):

```rst
:Overview: Counts the reads in a fastq file
:Input: FastQ raw data file
:Output:
    - count.txt

Usage
~~~~~~

::

    sequana init pipeline_count
    snakemake -s pipeline_count.rules -f

Requirements
~~~~~~~~~~~~~~

Here you should list the dependencies, which should match the file requirements.txt in ./sequana_pipelines/count/

(continues on next page)
Details
~~~~~~~~~~~~~

Rules and configuration details
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

count rule
~~~~~~~~~~~~~
.. snakemakerrule:: pipeline_count

Note: the README.rst uses Restructured syntax (not markdown)

Documenting the configuration file

The configuration should be in YAML format. You should comment top-level sections corresponding to a rule as follows:

```
# A block comment in docstring format
# This means a # character followed by a space and then
# the docstring. The first line made of ##### will be removed
# and is used to make the documentation clear. No spaces
# before the section (count:) here below.
#
count:
  item1: 1 # you can add comment for an item
  item2: 2 # you can add comment for an item
```

If valid, the block comment is interpreted and a tooltip will appear in Sequanix.

You can also use specific syntax to have specific widgets in Sequanix (see Sequanix Tutorial).

First, you may have a file browser widget by adding _file after a parameter:

```
# documentation here
#
count:
  reference_file:
```

You may also have the choice between several values, in which case you have to provide the different items inside the documentation as follows:

```
# documentation here
```

(continues on next page)
# adapter_choice__= ['PCRFree', 'TruSeq', None]
count:
    adapter_choice: PCRFree

Warning: Note the double underscore after _choice_. With this syntax, Sequanix will interpret the list and include the items in a dropdown button with 3 choices (PCRFree, TruSeq and None). This minimizes typo errors. You may need to add None if no selection is a valid choice.

Warning: note the = sign between _choice__ and the list of valide values

Further coding conventions

To print debugging information, warnings or more generally information, please do not use the print() function but the logger:

```python
from sequana import logger
logger.debug("test")
logger.info("test")
logger.warning("test")
logger.error("test")
logger.critical("test")
```

2.9.4 Testing with pytest

As a developer, when you change your code, you want to quickly test whether the modification(s) did not introduce any regression bugs. To do so, just type:

```
python setup.py test
```

Note: we moved from nosetests to pytest. This framework is slightly more flexible but the main reason to move was to be able to test Qt application. It appeared that it also has nice plugins such as multithreaded testing.

You will need to install pytest and some plugins. You can use conda to do so thanks to the requirements_dev.txt file:

```
conda install --list https://raw.githubusercontent.com/sequana/sequana/master/requirements_dev.yml
```

This command installs:
- pytest: main utility
- pytest-cov: coverage support
- pytest-qt: fixture for Qt
- pytest-xdist: allows multi threading
- pytest-mock: mocking feature
• pytest-timeout: report longest tests

For instance, you can use in the root directory of Sequana:

```bash
pytest -v --durations=10 test/ --cov=sequana --cov-report term-missing --timeout 300 -n 4
```

Here, -n 4 requires two CPUs to run the tests. The option durations=10 means "show the 10 longest tests".

We also adapt the setup.py and setup.cfg so that you can simply type:

```bash
python setup.py test
```

If you want to test a single file (e.g. test_pacbio):

```bash
cd test
pytest test_pacbio.py --cov sequana.pacbio --cov-report term-missing
```

## 2.9.5 Module reports

Sequana pipelines generate HTML reports. Those reports are created with the module reports stored in ./sequana/modules_report directory.

A module report creates one HTML page starting from a dataset generated by Sequana, or a known data structure. All modules reports inherit from `SequanaBaseModule` as shown hereafter. This class provides convenient methods to create the final HTML, which takes care of copying CSS and Javascript libraries.

To explain how to write a new module report, let us consider a simple example. We design here below a working example of a module report that takes as input a Pandas dataframe (a Pandas series made of a random normal distribution to be precise). The module report then creates an HTML page with two sections: a dynamic sortable table and a section with an embedded image. Each section is made of a dictionary that contains 3 keys:

- **name**: the HTML section name
- **anchor**: the ID HTML of the section
- **content**: a valid HTML code

First, you need to import the base class. Here we also import a convenient object called DataTable that will be used to create the sortable table in HTML using Javascript behind the scene.

```python
from sequana.modules_report.base_module import SequanaBaseModule
from sequana.utils.datatables_js import DataTable
```

Then, we define a new class called `MyModule` as follows:

```python
class MyModule(SequanaBaseModule):
```

followed by a constructor

```python
def __init__(self, df, output="mytest.html"):
    super().__init__()
    self.data = df
    self.summary = self.data.describe().to_frame()
    self.title = "Super Module"
    self.create_report_content()
    self.create_html(output)
```
This constructor stores the input argument (df) and computes some new data stored in the summary attribute. Here this computation is fast but in a real case example where computation may takes time, the computation should be performed outside of the module. We then store a title in the title attribute. Finally two methods are called. The first one creates the HTML sections (create_report_method); the second one (create_html) is inherited from SequanaBaseModule.

The first method is defined as follows:

```python
def create_report_content(self):
    self.sections = list()
    self.add_table()
    self.add_image()
```

Here, the method create_report_content() may be named as you wish but must define and fill the sections list (empty list is possible) with a set of HTML sections. In this example, we call two methods (add_table and add_image) that adds two HTML sections in the list. You may have as many add_ methods.

First, let us look at the add_table(). It creates an HTML section made of a dynamic HTML table based on the DataTable class. This class takes as input a Pandas DataFrame.

```python
def add_table(self):
    df = self.summary.copy()
    df.columns = ['data']
    df['url'] = ['http://sequana.readthedocs.org'] * len(df)
    table = DataTable(df, "table", index=True)
    table.datatables.datatables_options = {
        'scrollX': '300px',
        'pageLength': 15,
        'scrollCollapse': 'true',
        'dom': 'tB',
        "paging": "false",
        'buttons': ['copy', 'csv']}
    table.datatables.set_links_to_column('url', 'data')
    js = table.create_javascript_function()
    html_tab = table.create_datatable(float_format='%.3g')
    html = "{} {}".format(html_tab, js)
    self.sections.append({
        "name": "Table",
        "anchor": "table",
        "content": html
    })
```

Here, we first get some data (line 2) in the form of a Pandas time Series. We rename the column on line 3. This is a dataframe and the DataTable class takes as input Pandas dataframe that are then converted into flexible HTML table.

One nice feature about the DataTable is that we can add HTML links (URL) in a specific column of the data frame (line 3) and then link an existing column with this new URL column. This happens on line 11. The final HTML table will not show the URL column but the data column will be made of clickable cells.

The creation of the data table itself happens on line 5 to line 11 and line 12-14. There are two steps here: the creation of the HTML table itself (line 13) and the Javascript itself (line 12).

Once we have the HTML data, we can add it into the sections on line 16-19.

The second section is an HTML section with an image. It may be included with a standard approach (using the img
tag) but one can also use the `create_embedded_png()` method.

```python
def add_image(self):
    import pylab
    def plotter(filename):
        pylab.ioff()
        self.data.hist()
        pylab.savefig(filename)
    html = self.create_embedded_png(plotter, "filename",
        style='width:65%')
    self.sections.append({
        "name": "Image",
        "anchor": "table",
        "content": html
    })
```

Here is the full working example:

```python
from numpy import random
import pandas as pd

from sequana.modules_report.base_module import SequanaBaseModule
from sequana.utils.datatables_js import DataTable

class MyModule(SequanaBaseModule):
    def __init__(self, df, output="mytest.html"):
        super().__init__()
        self.data = df
        self.summary = self.data.describe().to_frame()

        self.title = "Super Module"
        self.create_report_content()
        self.create_html(output)

    def create_report_content(self):
        self.sections = list()
        self.add_table()
        self.add_image()

    def add_table(self):
        df = self.summary.copy()
        df.columns = ['data']
        df['url'] = ['http://sequana.readthedocs.org'] * len(df)

        table = DataTable(df, "table", index=True)
        table.datatable.datatable_options = {
            'scrollX': '300px',
            'pageLength': 15,
            'scrollCollapse': 'true',
            'dom': 'tB',
            'paging': 'false',
            'buttons': ['copy', 'csv']}
        table.datatable.set_links_to_column('url', 'data')
```

(continues on next page)
When using this module, one creates an HTML page called `mytest.html`. An instance of the page is available here: `report_example.html`

### 2.9.6 Documentation

If you add new code in the sequana library, please add documentation everywhere: in classes, functions, modules following docstring and sphinx syntax. To check that the documentation is correct, or to build the documentation locally, first install sphinx:

```
conda install sphinx sphinx_rtd_theme
```

and from the root directory of the source code:

```
cd doc
maje html
```
2.9.7 MultiQC

If you have several samples in a pipeline and the pipeline creates $N$ HTML reports and/or summary.json files thanks to the module report (see above), there is a high probability that you also want to have a multi summary.

We decided to use multiqc (http://multiqc.info/) for that purpose.

We consider the example used here above with the pipeline named pipeline_count. We suppose that the output is also made of a summary_count_SAMPLE.json file created for each sample. Let us assume you took care of creating a nice individual HTML pages (optional).

Now, you wish to create a multiQC report to summarize those individual sample analysis. This means you want to retrieve automatically the file sequana_summary_count_SAMPLE.json. Note that they may be named differently; for instance, sample/sequana_summary.json

In ./sequana/multiqc directory, add a file called pipeline_count.py

- Take as example the already existing file such as pacbio_qc.py
- update the sequana/multiqc/__init__.py to add the search pattern for your input (here summary_count*.json)
- update the sequana/multiqc/config.py to add the search pattern for your input (here summary_count*.json). This way, you can use "multiqc ." and sequana modules will use the pattern stored in config.py
- update the sequana/multiqc/multiqc_config.yaml to add the search pattern for your input (here summary_count*.json). This way, you can use a user define "multiqc . -c multiqc_config.yaml"
- In the setup.py, add the entry point following the example of pacbio_qc
- In the ./test/multiqc add a test in test_multiqc.py

To create the summary, we provide a convenient class in summary.Summary.

2.9.8 Singularity

We provide a Singularity file. It is in the main directory and must be kept there to be found by singularity-hub. Each commit to the Singularity file (in the master branch) will trigger this website to build a singularity image. The latest built image can be downloaded as follows:

```
singularity pull shub://sequana/sequana
```

Note that by default, this downloads the latest version. It is equivalent to adding a tag named "latest":

```
singularity pull shub://sequana/sequana:latest
```

In order to provide frozen built, you must use tags. This is achieved by adding extension to singularity files in the directory ./singularity. For example:

```
singularity/Singularity.0_6_2
```

will contain a recipe that fetches the version 0.6.2 of sequana on pypi. Once this file is created, the container is built on singularity hub and should never be changed again (except for bugs) !! Although you may also create a branch (e.g. named release_0_6_2), you still need to keep the singularity filename unique. Indeed, consider this case:

- branch master with a singularity/Singularity file
- branch release_0_6_2 with a singularity/Singularity file

Although those two files (if built on singularity) are in different branches, they will have the same URI (sequana/sequana:latest) so the latest will be considered and you have two identical containers.
So, whatever solution is chosen, a unique tag must always be added. We decided to only use the master branch for now. When downloading a container without the `--name` argument, your file is named:

```
sequana-sequana-{<release name>}_{<tag>}.simg
```

This may change in the future version of singularity. Once downloaded, use the container as follows:

**Note:** only Singularity files that have been changed since the last commit will be built with Automatic Building in this fashion. Empty commits won’t work.

## 2.10 Rules

As of August 2017, Sequana has about 80 different rules. The list is available from the source code. We design our rules following some strict conventions as explained in the Developer guide section.

Rules are documented and we developed a Sphinx extension to automatically add their docstring in this documentation. For example, the documentation of the rule `fastq_sampling` looks like:

In order to use a Sequana rule in your pipeline, add this code:

```python
from sequana_pipetools import snaketools as sm
include: sm.modules["fastq_sampling"]
```

This takes care of the physical location of the rule. Of course, you will then need to look at the documentation and define the required variables in your pipeline. For instance, in the example above, given the documentation, you will need to define those two variables:

```
__fastq_sampling_input_fastq
__fastq_sampling_output_fastq
```

and have a configuration file with:

```
fastq_sampling:
  N: 1000
```

Many rules are used inside the Sequana pipelines but not all. For instance, the codecs rules (e.g. gz_to_bzip) are used in standalones.

Please see the Pipelines section for other rule documentation (e.g. bwa, fastqc, ...).

## 2.11 Wrappers

As of August 2021, Sequana team created the sequana wrappers repository, which is intended to replace the rules. The advantage is that wrappers can be tested with a continuous integration.

Wrappers are used within a Snakemake rule. When you call your Snakemake pipeline, you will need to add:

```
--wrapper-prefix git+file:https://github.com/sequana/sequana-wrappers/
```

We provide documentation for each wrapper. It can be included in this documentation thanks to a sphinx extension. For example:
Here is a non exhaustive list of documented wrappers.

### 2.11.1 fastqc

### 2.11.2 rulegraph

### 2.11.3 bowtie2

### 2.12 References

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>References</td>
</tr>
<tr>
<td>- Assembly related</td>
</tr>
<tr>
<td>- BAMTOOLS related</td>
</tr>
<tr>
<td>- Coverage (bedtools module)</td>
</tr>
<tr>
<td>- CIGAR tools</td>
</tr>
<tr>
<td>- Coverage (theoretical)</td>
</tr>
<tr>
<td>- Access to online database (e.g. ENA)</td>
</tr>
<tr>
<td>- Enrichment</td>
</tr>
<tr>
<td>- Experimental design</td>
</tr>
<tr>
<td>- FASTQ module</td>
</tr>
<tr>
<td>- FASTA module</td>
</tr>
<tr>
<td>- Feature counts module</td>
</tr>
<tr>
<td>- Sequence module</td>
</tr>
<tr>
<td>- Kmer module</td>
</tr>
<tr>
<td>- Taxonomy related (Kraken - Krona)</td>
</tr>
<tr>
<td>- Pacbio module</td>
</tr>
<tr>
<td>- Phred quality</td>
</tr>
<tr>
<td>- RiboDesigner</td>
</tr>
<tr>
<td>- RNAdiff</td>
</tr>
<tr>
<td>- Running median</td>
</tr>
<tr>
<td>- Snpeff module</td>
</tr>
<tr>
<td>- General tools</td>
</tr>
<tr>
<td>- Format IO</td>
</tr>
<tr>
<td>- VCF module</td>
</tr>
</tbody>
</table>
2.12.1 Assembly related

class BUSCO(filename='full_table_testbusco.tsv')
Wrapper of the BUSCO output

"BUSCO provides a quantitative measures for the assessment of a genome assembly, gene set, transcriptome completeness, based on evolutionarily-informed expectations of gene content from near-universal single-copy orthologs selected from OrthoDB v9." -- BUSCO website 2017

This class reads the full report generated by BUSCO and provides some visualisation of this report. The information is stored in a dataframe df. The score can be retrieved with the attribute score in percentage in the range 0-100.

Reference  http://busco.ezlab.org/

**constructor**

**Filename** a valid BUSCO input file (full table). See example in sequana code source (testing)

**get_summary_string()**

**pie_plot(filename=None, hold=False)**

Plot PIE plot of the status (complete / fragment / missed)

```python
from sequana import BUSCO, sequana_data
b = BUSCO(sequana_data("test_busco_full_table.tsv"))
b.pie_plot()
```

**scatter_plot(filename=None, hold=False)**

Scatter plot of the score versus length of each ortholog

```python
from sequana import BUSCO, sequana_data
b = BUSCO(sequana_data("test_busco_full_table.tsv"))
b.scatter_plot()
```

Missing are not show since there is no information about contig.

**property score**

**summary()**

Return summary information of the missing, completed, fragmented orthologs

class Contigs(filename, reference=None, bamfile=None, mode='canu')

```bash
minimap2 -x map-pb reference filename -a > temp.sam bioconvert sam2bam temp.sam temp.bam
```

**bar_plot_contigs_length()**

**get_contig_per_chromosome()**
get_df(window=100)

hist_plot_contig_length(bins=40, fontsize=16)
    Plot distribution of contig lengths

plot_contig_length_vs_GC()

plot_contig_length_vs_nreads(fontsize=16)

plot_scatter_contig_length_nread_cov(fontsize=16, vmin=0, vmax=50, min_nreads=20, min_length=50000)

stats()

class ContigsBase(filename)
    get_gc(window=100)
    plot_contig_length_vs_GC(alpha=0.5)
    scatter_length_cov_gc(min_length=200, min_cov=10)

class ContigsSpades(filename)
    hist_contig_length(bins=30, fontsize=16)

2.12.2 BAMTOOLS related

Tools to manipulate BAM/SAM files

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment</td>
<td>Helper class to retrieve info about Alignment</td>
</tr>
<tr>
<td>BAM(filename, *args)</td>
<td>BAM reader.</td>
</tr>
<tr>
<td>CRAM(filename, *args)</td>
<td>CRAM Reader.</td>
</tr>
<tr>
<td>MultiBAM()</td>
<td>Convenient structure to store several BAM files</td>
</tr>
<tr>
<td>SAM(filename, *args)</td>
<td>SAM Reader.</td>
</tr>
<tr>
<td>SAMFlags([value])</td>
<td>Utility to extract bits from a SAM flag</td>
</tr>
<tr>
<td>SAMBAMbase(filename[, mode])</td>
<td>Base class for SAM/BAM/CRAM data sets</td>
</tr>
</tbody>
</table>

Note: BAM being the compressed version of SAM files, we do not implement any functionalities related to SAM files. We strongly encourage developers to convert their SAM to BAM.

class Alignment(alignment)
    Helper class to retrieve info about Alignment
    Takes an alignment as read by BAM and provides a simplified version of pysam.Alignment class.

>>> from sequana.bamtools import Alignment
>>> from sequana import BAM, sequana_data
>>> b = BAM(sequana_data("test.bam"))
>>> segment = next(b)
>>> align = Alignment(segment)
>>> align.as_dict()
>>> align.FLAG
353
The original data is stored in hidden attribute `_data` and the following values are available as attributes or dictionary:

- **QNAME**: a query template name. Reads/segment having same QNAME come from the same template. A QNAME set to `*` indicates the information is unavailable. In a sam file, a read may occupy multiple alignment
- **FLAG**: combination of bitwise flags. See `SAMFlags`
- **RNAME**: reference sequence
- **POS**
- **MAPQ**: mapping quality if segment is mapped. equals -10 log10 Pr
- **CIGAR**: See `sequana.cigar.Cigar`
- **RNEXT**: reference sequence name of the primary alignment of the NEXT read in the template
- **PNEXT**: position of primary alignment
- **TLEN**: signed observed template length
- **SEQ**: segment sequence
- **QUAL**: ascii of base quality

**constructor**

Parameters **alignment** -- alignment instance from `BAM`

```python
>>> from sequana import CS
>>> CS('-a:6-g:14+g:2+c:9\'ac:10-a:13-a')
{'D': 3, 'I': 2, 'M': 54, 'S': 1}
```

When using some mapper, CIGAR are stored in another format called CS, which also includes the substitutions. See minimap2 documentation for details.

**class** `SAM(filename, *args)`

SAM Reader. See `SAMBAMbase` for details

**class** `SAMBAMbase(filename, mode='r', *args)`

Base class for SAM/BAM/CRAM data sets

We provide a few test files in Sequana, which can be retrieved with `sequana_data:`
>>> from sequana import BAM, sequana_data
>>> b = BAM(sequana_data("test.bam"))
>>> len(b)
1000
>>> from sequana import CRAM
>>> b = CRAM(sequana_data("test_measles.cram"))
>>> len(b)
60

bam_analysis_to_json(filename)
Create a json file with information related to the bam file.
This includes some metrics (see get_stats(); eg MAPQ), combination of flags, SAM flags, counters
about the read length.

boxplot_qualities(max_sample=500000)
Same as in sequana.fastq.FastQC

get_df(max_align=-1)

get_df_concordance(max_align=-1)
This methods returns a dataframe with Insert, Deletion, Match, Substitution, read length, concordance (see
below for a definition)
Be aware that the SAM or BAM file must be created using minimap2 and the --cs option to store the CIGAR
in a new CS format, which also contains the information about substitution. Other mapper are also handled
(e.g. bwa) but the substitution are solely based on the NM tag if it exists.
alignment that have no CS tag or CIGAR are ignored.

get_estimate_insert_size(max_entries=100000, upper_bound=1000, lower_bound=-1000)

Here we show that about 3000 alignments are enough to get a good estimate of the insert size.

get_flags_as_df()
Returns decomposed flags as a dataframe

2.12. References
See also:

**SAMFlags** for meaning of each flag

**get_gc_content()**

Return GC content for all reads (mapped or not)

See also:

**plot_gc_content()**

**get_length_count()**

Return counter of all fragment lengths

**get_mapped_read_length()**

Return dataframe with read length for each read

**get_mapq_as_df**(max_entries=-1)

Return dataframe with mapq for each read

**get_read_names()**

Return the reads’ names

**get_samflags_count()**

Count how many reads have each flag of SAM format.

Returns dictionary with keys as SAM flags

**get_samtools_stats_as_df()**

Return a dictionary with full stats about the BAM/SAM file

The index of the dataframe contains the flags. The column contains the counts.
Note: uses samtools behind the scene

get_stats()
Return basic stats about the reads

Returns
dictionary with basic stats:
- total_reads: number reads ignoring supplementaty and secondary reads
- mapped_reads: number of mapped reads
- unmapped_reads: number of unmapped
- mapped_proper_pair: R1 and R2 mapped face to face
- reads_duplicated: number of reads duplicated

Warning: works only for BAM files. Use get_samtools_stats_as_df() for SAM files.

get_stats_full(mapq=30, max_entries=-1)
hist_coverage(chrom=None, bins=100)

from sequana import sequana_data, BAM
b = BAM(sequana_data("measles.fa.sorted.bam"))
b.hist_coverage()

hist_soft_clipping()
histogram of soft clipping length ignoring supplementary and secondary reads

infer_strandness(reference_bed, max_entries, mapq=30)

Parameters
- reference_bed -- a BED file (12-columns with columns 1,2,3,6 used) or GFF file (column 1, 3, 4, 5, 6 are used)
- mapq -- ignore alignment with mapq below 30.
- max_entries -- can be long. max_entries restrict the estimate

Strandness of transcript is determined from annotation while strandness of reads is determined from alignments.
For non strand-specific RNA-seq data, strandness of reads and strandness of transcript are independent.
For strand-specific RNA-seq data, strandness of reads is determined by strandness of transcripts.
This functions returns a list of 4 values. First one indicates whether data is paired or not. Second and third one are ratio of reads explained by two types of strandness of reads vs transcripts. Last values are fractions of reads that could not be explained. The values 2 and 3 tell you whether this is a strand-specific dataset.
If similar, it is no strand-specific. If the first value is close to 1 while the other is close to 0, this is a strand-specific dataset

property is_paired
**property is_sorted**
return True if the BAM is sorted

**mRNA_inner_distance**(*refbed*, *low_bound=-250, up_bound=250, step=5, sample_size=1000000, q_cut=30*)
Estimate the inner distance of mRNA pair end fragment.

```python
from sequana import BAM, sequana_data
b = BAM(sequana_data("test_hg38_chr18.bam"))
df = b.mRNA_inner_distance(sequana_data("hg38_chr18.bed"))
```

**plot_bar_flags**(log=True, fontsize=16, filename=None)
Plot an histogram of the flags contained in the BAM

```python
from sequana import BAM, sequana_data
b = BAM(sequana_data('test.bam'))
b.plot_bar_flags()
```

See also:
**SAMFlags** for meaning of each flag

**plot_bar_mapq**(fontsize=16, filename=None)
Plots bar plots of the MAPQ (quality) of alignments

```python
from sequana import BAM, sequana_data
b = BAM(sequana_data('test.bam'))
b.plot_bar_mapq()
```

**plot_coverage**(chrom=None)
Please use GenomeCov for more sophisticated tools to plot the genome coverage

```python
from sequana import sequana_data, BAM
b = BAM(sequana_data('measles.fa.sorted.bam'))
b.plot_coverage()
```

**plot_gc_content**(fontsize=16, ec='k', bins=100)
plot GC content histogram

**Params**
- **bins** a value for the number of bins or an array (with a copy() method)

**Parameters**
- **ec** -- add black contour on the bars

```python
from sequana import sequana_data
b = BAM(sequana_data('test.bam'))
b.plot_gc_content()
```

**plot_indel_dist**(fontsize=16)
Plot indel count (+ ratio)

**Return** list of insertions, deletions and ratio insertion/deletion for different length starting at 1

```python
from sequana import sequana_data
b = BAM(sequana_data('measles.fa.sorted.bam'))
b.plot_indel_dist()
```
What you see on this figure is the presence of 10 insertions of length 1, 1 insertion of length 2 and 3 deletions of length 1.

# Note that in samtools, several insertions or deletions in a single alignment are ignored and only the first one seems to be reported. For instance 10M1I10M1I stored only 1 insertion in its report. Same comment for deletions.

**Todo:** speed up and handle long reads cases more efficiently by storing INDELS as histograms rather than lists.

```python
plot_insert_size(max_entries=100000, bins=100, upper_bound=1000, lower_bound=-1000, absolute=False)
```

This gives an idea of the insert size without taking into account any intronic gap. The mode should give a good idea of the insert size though.

```python
plot_read_length()
```

Plot occurrences of aligned read lengths

```python
from sequana import sequana_data, BAM
b = BAM(sequana_data("test.bam"))
b.plot_read_length()
```

**reset()**

**property summary**

Count flags/mapq/read length in one pass.

```python
property summary
```

**to_fastq(filename)**

Export the BAM to FastQ format

```python
Todo: comments from original reads are not in the BAM so will be missing
```

Method 1 (bedtools):

```bash
bedtools bamtofastq -i JB409847.bam -fq test1.fastq
```

Method 2 (samtools):

```bash
samtools bam2fq JB409847.bam > test2.fastq
```

Method 3 (sequana):

```python
from sequana import BAM
BAM(filename)
BAM.to_fastq("test3.fastq")
```

Note that the samtools method removes duplicated reads so the output is not identical to method 1 or 3.

```python
class SAMFlags(value=4095)
Utility to extract bits from a SAM flag
```

```python
>>> from sequana import SAMFlags
>>> sf = SAMFlags(257)
```

(continues on next page)
>>> sf.get_flags()
[1, 256]

You can also print the bits and their description:

```python
print(sf)
```

<table>
<thead>
<tr>
<th>bit</th>
<th>Meaning/description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>mapped segment</td>
</tr>
<tr>
<td>1</td>
<td>template having multiple segments in sequencing</td>
</tr>
<tr>
<td>2</td>
<td>each segment properly aligned according to the aligner</td>
</tr>
<tr>
<td>4</td>
<td>segment unmapped</td>
</tr>
<tr>
<td>8</td>
<td>next segment in the template unmapped</td>
</tr>
<tr>
<td>16</td>
<td>SEQ being reverse complemented</td>
</tr>
<tr>
<td>32</td>
<td>SEQ of the next segment in the template being reverse complemented</td>
</tr>
<tr>
<td>64</td>
<td>the first segment in the template</td>
</tr>
<tr>
<td>128</td>
<td>the last segment in the template</td>
</tr>
<tr>
<td>256</td>
<td>secondary alignment</td>
</tr>
<tr>
<td>512</td>
<td>not passing filters, such as platform/vendor quality controls</td>
</tr>
<tr>
<td>1024</td>
<td>PCR or optical duplicate</td>
</tr>
<tr>
<td>2048</td>
<td>supplementary alignment</td>
</tr>
</tbody>
</table>


**get_flags()**
- Return the individual bits included in the flag

**get_meaning()**
- Return all description sorted by bit

## 2.12.3 Coverage (bedtools module)

Utilities for the genome coverage

**class ChromosomeCov(genomecov, chrom_name, thresholds=None, chunksize=5000000)**

Factory to manipulate coverage and extract region of interests.

Example:

```python
from sequana import GenomeCov, sequana_data
filename = sequana_data("virus.bed")

gencov = GenomeCov(filename)

chrcov = gencov[0]
chrcov.running_median(n=3001)
chrcov.compute_zscore()
chrcov.plot_coverage()

df = chrcov.get_rois().get_high_rois()
```
The \textit{df} variable contains a dataframe with high region of interests (over covered)
If the data is large, the input data set is split into chunk. See \texttt{chunksize}, which is 5,000,000 by default.
If your data is larger, then you should use the \texttt{run()} method.

\textbf{See also:}
sequana\_coverage standalone application

\textbf{constructor}

\textbf{Parameters}

- \texttt{df} -- dataframe with position for a chromosome used within \texttt{GenomeCov}. Must contain the following columns: ["pos", "cov"]
- \texttt{genomedb} --
- \texttt{chrom\_name} -- to save space, no need to store the chrom name in the dataframe.
- \texttt{thresholds} -- a data structure \texttt{DoubleThresholds} that holds the double threshold values.
- \texttt{chunksize} -- if the data is large, it is split and analysed by chunk. In such situations, you should use the \texttt{run()} instead of calling the running\_median and compute\_zscore functions.

\textbf{property BOC}

\textbf{breadth of coverage}

\textbf{property C3}

\textbf{property C4}

\textbf{property CV}

The coefficient of variation (CV) is defined as \(\sigma / \mu\)

\textbf{property DOC}

\textbf{depth of coverage}

\textbf{property STD}

\textbf{standard deviation of depth of coverage}

\textbf{property bed}

\textbf{compute\_zscore}(\textit{k}=2, \textit{use\_em}=True, \textit{clip}=4, \textit{verbose}=True)

Compute zscore of coverage and normalized coverage.

\textbf{Parameters}

- \texttt{k (int)} -- Number gaussian predicted in mixture (default = 2)
- \texttt{clip (float)} -- ignore values above the clip threshold

Store the results in the \textit{df} attribute (dataframe) with a column named \texttt{zscore}.

\textbf{Note:} needs to call \texttt{running\_median()} before hand.

\textbf{property df}
property evenness

Return Evenness of the coverage

Reference Konrad Oexle, Journal of Human Genetics 2016, Evaluation of the evenness score in NGS.

work before or after normalisation but lead to different results.

get_centralness(threshold=3)

Proportion of central (normal) genome coverage

This is 1 - (number of non normal data) / (total length)

Note: depends on the thresholds attribute being used.

Note: depends slightly on \(W\) the running median window

get_gaussians()

gc_correlation()

Return the correlation between the coverage and GC content

The GC content is the one computed in GenomeCov.compute_gc_content() (default window size is 101)

g_max_g_c_correlation(reference, guess=100)

Plot correlation between coverage and GC content by varying the GC window

The GC content uses a moving window of size \(W\). This parameter affects the correlation between coverage and GC. This function find the optimal window length.

g_rois()

Keep positions with zscore outside of the thresholds range.

Returns a dataframe from FilteredGenomeCov

Note: depends on the thresholds low and high values.

g_size()

g_stats()

Return basic stats about the coverage data

only "cov" column is required.

Returns dictionary

g_summary(C3=None, C4=None, stats=None, caller='sequana.bedtools')

moving_average(n, circular=False)

Compute moving average of the genome coverage

Parameters

- \(n\) -- window's size. Must be odd
- \(circular\) (bool) -- is the chromosome circular or not

2.12. References
Store the results in the `df` attribute (dataframe) with a column named `ma`.

```
next()
```

```
plot_coverage(filename=None, fontsize=16, rm_lw=1, rm_color='#0099cc',
             rm_label='Running median',
             th_lw=1, th_color='r', th_ls='--', main_color='k', main_lw=1,
             main_kwargs={},
             sample=True, set_ylimits=True, x1=None, x2=None, clf=True)
```

Plot coverage as a function of base position.

**Parameters**

- `filename` --
- `rm_lw` -- line width of the running median
- `rm_color` -- line color of the running median
- `rm_color` -- label for the running median
- `th_lw` -- line width of the thresholds
- `th_color` -- line color of the thresholds
- `main_color` -- line color of the coverage
- `main_lw` -- line width of the coverage
- `sample` -- if there are more than 1,000,000 points, we use an integer step to skip data points. We can still plot all points at your own risk by setting this option to False
- `set_ylimits` -- we want to focus on the "normal" coverage ignoring unusual excess. To do so, we set the yaxis range between 0 and a maximum value. This maximum value is set to the minimum between the 10 times the mean coverage and 1.5 the maximum of the high coverage threshold curve. If you want to let the ylimits free, set this argument to False
- `x1` -- restrict lower x value to `x1`
- `x2` -- restrict lower x value to `x2` (`x2` must be greater than `x1`)

**Note:** if there are more than 1,000,000 points, we show only 1,000,000 by points. For instance for 5,000,000 points,

```
plot_gc_vs_coverage(filename=None, bins=None, Nlevels=None, fontsize=20,
                    norm='log', ymin=0, ymax=100, contour=True, cmap='BrBG', **kwargs)
```

Plot histogram 2D of the GC content versus coverage

```
plot_hist_coverage(logx=True, logy=True, fontsize=16, N=25, fignum=1, hold=False,
                   alpha=0.8, ec='k',
                   filename=None, zorder=10, **kw_hist)
```

**Parameters**

- `N` --
- `ec` --
plot_hist_normalized_coverage(filename=None, binwidth=0.05, max_z=3)

Barplot of the normalized coverage with gaussian fitting

plot_hist_zscore(fontsize=16, filename=None, max_z=6, binwidth=0.5, **hist_kargs)

Barplot of the zscore values

plot_rois(x1, x2, set_ylimits=False, rois=None, fontsize=16, color_high='r', color_low='g', clf=True)

reset()

property rois

run(W, k=2, circular=False, binning=None, cnv_delta=None)

running_median(n, circular=False)

Compute running median of genome coverage

Parameters

• n (int) -- window's size.

• circular (bool) -- if a mapping is circular (e.g. bacteria whole genome sequencing), set to True

Store the results in the df attribute (dataframe) with a column named rm.

Changed in version 0.1.21: Use Pandas rolling function to speed up computation.

thresholds

try: self.thresholds = thresholds.copy()
except: self.thresholds = DoubleThresholds()

to_csv(filename=None, start=None, stop=None, **kwargs)

Write CSV file of the dataframe.

Parameters

• filename (str) -- csv output filename. If None, return string.

• start (int) -- start row index.

• stop (int) -- stop row index.

Params of pandas.DataFrame.to_csv():

Parameters

• columns (list) -- columns you want to write.

• header (bool) -- determine if the header is written.

• index (bool) -- determine if the index is written.

• float_format (str) -- determine the float format.

class DoubleThresholds(low=-3, high=3, ldtr=0.5, hdtr=0.5)

Simple structure to handle the double threshold for negative and positive sides

Used yb GenomeCov and related classes.

dt = DoubleThresholds(-3, 0.5, 0.5)
This means the low threshold is -3 while the high threshold is 4. The two following values must be between 0 and 1 and are used to define the value of the double threshold set to half the value of the main threshold.

Internally, the main thresholds are stored in the low and high attributes. The secondary thresholds are derived from the main thresholds and the two ratios. The ratios are named ldtr and hdtr for low double threshold ratio and high double threshold ration. The secondary thresholds are denoted low2 and high2 are are update automatically if low, high, ldtr or hdtr are changed.

```python
from sequana import GenomeCov, sequana_data

filename = sequana_data('JB409847.bed')
reference = sequana_data('JB409847.fasta')

gencov = GenomeCov(filename)

# you can change the thresholds:
gencov.thresholds.low = -4

result = gencov.compute_gc_content(reference)

gencov = GenomeCov(filename)
for chrom in gencov:
    chrom.running_median(n=3001, circular=True)
    chrom.compute_zscore()
    chrom.plot_coverage()

result = gencov[0].plot_coverage()
```

Results are stored in a list of `ChromosomeCov` named `chr_list`. For Prokaryotes and small genomes, this API is convenient but takes lots of memory for larger genomes.

Computational time information: scanning 24,000,000 rows

- constructor (scanning 40,000,000 rows): 45s
• select contig of 24,000,000 rows: 1min20
• running median: 16s
• compute zscore: 9s
• c.get_rois() :

**constructor**

Parameters

- **input_filename** (`str`) -- the input data with results of a bedtools genomcov run. This is just a 3-column file. The first column is a string (chromosome), second column is the base position and third is the coverage.
- **genbank_file** (`str`) -- annotation file of your reference.
- **low_threshold** (`float`) -- threshold used to identify under-covered genomic region of interest (ROI). Must be negative
- **high_threshold** (`float`) -- threshold used to identify over-covered genomic region of interest (ROI). Must be positive
- **ldtr** (`float`) -- fraction of the low_threshold to be used to define the intermediate threshold in the double threshold method. Must be between 0 and 1.
- **rdtr** (`float`) -- fraction of the low_threshold to be used to define the intermediate threshold in the double threshold method. Must be between 0 and 1.
- **chunksize** -- size of segments to analyse. If a chromosome is larger than the chunk size, it is split into N chunks. The segments are analysed independently and ROIs and summary joined together. Note that GC, plotting functionalities will only plot the first chunk.
- **force** -- some constraints are set in the code to prevent unwanted memory issues with specific data sets of parameters. Currently, by default, (i) you cannot set the threshold below 2.5 (considered as noise).
- **chromosome_list** -- list of chromosomes to consider (names). This is useful for very large input data files (hundreds million of lines) where each chromosome can be analysed one by one. Used by the sequana_coverage standalone. The only advantage is to speed up the constructor creation and could also be used by the Snakemake implementation.

**property circular**

Get the circularity of chromosome(s). It must be a boolean.

**compute_gc_content** (`fasta_file`, `window_size`=101, `circular`=False, `letters`=['G', 'C', 'c', 'g'])

Compute GC content of genome sequence.

Parameters

- **fasta_file** (`str`) -- fasta file name.
- **window_size** (`int`) -- size of the sliding window.
- **circular** (`bool`) -- if the genome is circular (like bacteria chromosome)

Store the results in the `ChromosomeCov.df` attribute (dataframe) with a column named gc.
property \texttt{feature\_dict}

Get the features dictionary of the genbank.

property \texttt{gc\_window\_size}

Get or set the window size to compute the GC content.

property \texttt{genbank\_filename}

Get or set the genbank filename to annotate ROI detected with \texttt{ChromosomeCov.get\_roi()}. Changing the genbank filename will configure the \texttt{GenomeCov.feature\_dict}.

\texttt{get\_stats()}

Return basic statistics for each chromosome

\textbf{Returns} dictionary with chromosome names as keys and statistics as values.

\textbf{See also:}

\texttt{ChromosomeCov}.

\textbf{Note:} used in \texttt{sequana\_summary} standalone

\texttt{hist(logx=True, logy=True, fignum=1, N=25, lw=2, **kwargs)}

\texttt{to\_csv(output\_filename, **kwargs)}

Write all data in a csv.

\textbf{Parameters}

- \texttt{output\_filename (str)} -- csv output file name.
- \texttt{kwargs (dict)} -- parameters of pandas.DataFrame.to_csv().

property \texttt{window\_size}

Get or set the window size to compute the running median. Size must be an integer.

### 2.12.4 CIGAR tools

\texttt{class Cigar(cigarstring)}

```
>>> from sequana.cigar import Cigar
>>> c = Cigar("2S30M1I")
>>> len(c)
33

>>> c = Cigar("1S1S1S1S")
>>> c.compress()
>>> c.cigarstring
'4S'
```

Possible CIGAR types are:

- "M" for alignment MATCH (0)
- "I" for Insertion to the reference (1)
- "D" for deletion from the reference 2
- "N" for skipped region from the reference 3
• "S" for soft clipping (clipped sequence present in seq) 4
• "H" for hard clipping (clipped sequence NOT present in seq) 5
• "P" for padding (silent deletion from padded reference)
• "=" for equal
• "X" for diff (sequence mismatched)
• "B" for back !!!! could be also NM ???

!!! BWA MEM get_cigar_stats returns list with 11 items Last item is !!! what is the difference between M and =

Note: the length of the query sequence based on the CIGAR is calculated by adding the M, I, S, =, or X and other operations are ignored. source: https://stackoverflow.com/questions/39710796/infer-the-length-of-a-sequence-using-the-cigar/39812985#39812985

Reference https://github.com/samtools/htslib/blob/develop/htslib/sam.h

Constructor

Parameters cigarstring (str) -- the CIGAR string.

Note: the input CIGAR string validity is not checked. If an unknown type is found, it is ignored generally. For instance, the length of 1S100Y is 1 since Y is not correct.

as_dict()

Return cigar types and their count

Returns dictionary

Note that repeated types are added:

```python
>>> c = Cigar('1S2M1S')
```

```python
>> c.as_dict()
{'S':2,'M':2}
```

as_sequence()

as_tuple()

Decompose the cigar string into tuples keeping track of repeated types

Returns tuple

```python
>>> from sequana import Cigar
```

```python
>>> c = Cigar("1S2M1S")
```

```python
>> c.as_tuple()
(("S", 1), ("M", 2), ("S", 1))
```

cigarstring

the CIGAR string attribute
compress()
    1S1S should become 2S. inplace modification

pattern = '\(\d+\)([A-Za-z])?\)

stats()
    Returns number of occurrence for each type found in types

>>> c = Cigar("1S2M1S")
>>> c.stats()
[2, 0, 0, 0, 2, 0, 0, 0, 0, 0]

types = 'MIDNSHP=XB'
    valid CIGAR types

fetch_clip(chrom, start, cigar)
fetch_deletion(chrom, start, cigar)
fetch_exon(chrom, start, cigar)
fetch_insertion(chrom, start, cigar)
fetch_intron(chrom, start, cigar)

2.12.5 Coverage (theoretical)

class Coverage(N=None, L=None, G=None, a=None)
    Utilities related to Lander and Waterman theory

    We denote $G$ the genome length in nucleotides and $L$ the read length in nucleotides. These two numbers are in
    principle well defined since $G$ is defined by biology and $L$ by the sequencing machine.

    The total number of reads sequenced during an experiment is denoted $N$. Therefore the total number of nu-
    cleotides is simply $NL$.

    The depth of coverage (DOC) at a given nucleotide position is the number of times that a nucleotide is covered
    by a mapped read.

    The theoretical fold-coverage is defined as :

    \[
    a = \frac{NL}{G}
    \]

    that is the average number of times each nucleotide is expected to be sequenced (in the whole genome). The
    fold-coverage is often denoted $aX$ (e.g., 50X).

    In the Coverage class, $G$ and $N$ are fixed at the beginning. Then, if one changes $a$, then $N$ is updated and
    vice-versa so that the relation $a = NL/G$ is always true:

>>> cover = Coverage(G=1000000, L=100)
>>> cover.N = 1000000  # number of reads
>>> cover.a            # What is the mean coverage
10
>>> cover.a = 50
>>> cover.N
5000000
From the equation aforementioned, and assuming the reads are uniformly distributed, we can answer a few interesting questions using probabilities.

In each chromosome, a read of length $L$ could start at any position (except the last position $L-1$). So in a genome $G$ with $n_c$ chromosomes, there are $G - n_c(L - 1)$ possible starting positions. In general $G >> n_c(L - 1)$ so the probability that one of the $N$ read starts at any specific nucleotide is $N/G$.

The probability that a read (of length $L$) covers a given position is $L/G$. The probability of not covering that location is $1 - L/G$. For $N$ fragments, we obtain the probability $(1 - L/G)^N$. So, the probability of covering a given location with at least one read is:

$$P = 1 - \left(1 - \frac{L}{G}\right)^N$$

Since in general, $N >> 1$, we have:

$$P = 1 - \exp^{-NL/G}$$

From this equation, we can derive the fold-coverage required to have e.g., $E = 99\%$ of the genome covered:

$$a = \log\left(-\frac{1}{(E - 1)}\right)$$

equivalent to

$$a = -\log(1 - E)$$

The method `get_required_coverage()` uses this equation. However, for numerical reason, one should not provide $E$ as an argument but $(1-E)$. See `get_required_coverage()`

Other information can also be derived using the methods `get_mean_number_contigs()`, `get_mean_contig_length()`, `get_mean_contig_length()`.

See also:

`get_table()` that provides a summary of all these quantities for a range of coverage.


**property G**

- genome length

**property L**

- length of the reads

**property N**

- number of reads defined as $aG/L$

**property a**

- coverage defined as $NL/G$

**get_mean_contig_length()**

- Expected length of the contigs

$$\frac{e^a - 1) L}{a}$$
get_mean_number_contigs()
Expected number of contigs
A binomial distribution with parameters $N$ and $p$

$$\binom{aG}{L} \exp^{-a}$$

get_mean_reads_per_contig()
Expected number of reads per contig
Number of reads divided by expected number of contigs:

$$\frac{N}{N \exp^{-a}} = e^a$$

get_percent_genome_sequenced()
Return percent of the genome covered

$$100(1 - \exp^{-a})$$

get_required_coverage($M=0.01$)
Return the required coverage to ensure the genome is covered
A general question is what should be the coverage to make sure that e.g. $E=99\%$ of the genome is covered by at least a read.
The answer is:

$$\log^{-1/(E-1)}$$

This equation is correct but have a limitation due to floating precision. If one provides $E=0.99$, the answer is 4.6 but we are limited to a maximum coverage of about 36 when one provides $E=0.9999999999999999$ after which $E$ is rounded to 1 on most computers. Besides, it is no convenient to enter all those numbers. A scientific notation would be better but requires to work with $M = 1 - E$ instead of $E$.

$$\log^{-1/M}$$

So instead of asking the question what is the requested fold coverage to have 99% of the genome covered, we ask the question what is the requested fold coverage to have 1% of the genome not covered. This allows us to use $M$ values as low as 1e-300 that is a fold coverage as high as 690.

Parameters $M$ (float) -- this is the fraction of the genome not covered by any reads (e.g. 0.01 for 1%). See note above.

Returns the required fold coverage

# The inverse equation is required fold coverage = [log(-1/(E - 1))]

get_summary()
Return a summary (dictionary) for the current fold coverage

get_table($coverage=None$
Return a summary dataframe for a set of fold coverage

Parameters $coverage$ (list) -- if None, coverage list starts at 0.5 and ends at 10 with a step of 0.5
Uncovered genome

Required coverage

Uncovered genome
2.12.6 Access to online database (e.g. ENA)

Utilities to access to online FASTA, taxon, lineage ...

**class ENADownload**
Downloader to retrieve genome fasta files from ENA amongst other things

In order to facilitate the download of FASTA files (e.g. to build a Kraken DB), this class can be used to download a bunch of FASTA files, or just one given its accession.

Some **OLD** pre-defined lists are available from ENA. We refer to them as **virus**, **plasmid**, **phage**, **archaealvirus**, **archaea**, **bacteria**, **organelle**, **viroid**.

**Warning:** the header of the FASTA files are changed to add the GI number instead of embl so that it can be used by our kraken builder class.

**constructor**

**add_gi_to_header(acc)**
Kraken will only accept the GI from NCBI so we need to convert the ENA accession to GI numbers

**download_fasta(filelist, output_dir=None)**
Download a FASTA (or list of)

Parameters **filelist** -- a name to find on the ENA web server OR the name of an accession number or a file with accession numbers (1 column)

**Warning:** The filename is named after the accession without .X number If there are several variant .1, .2 the later will be used. This should not happen if the list is properly defined.

**ena_id_to_gi_number(identifiers)**

**class EUtilsTools**
Utilities to fetch information about accession numbers

```python
>>> from sequana.databases import EUtilsTools
>>> et = EUtilsTools()
>>> et.accession_to_info("K01711.1")
{'K01711.1': {'accession': '331784',
'comment': 'Measles virus (strain Edmonston), complete genome',
'gi': '331784',
'identifier': 'gi|331784|gb|K01711.1|MEANPCG[331784]|',
'taxid': '11234'}}
```

**accession_to_info(ids)**
An accession or list of them returns list of dictionaries

**get_fasta(accession)**

**class NCBIDownload**

**download_assembly_report(category, output=None)**
download_genomes_from_ncbi\(category, \text{email}=\text{'sequana@pasteur.fr'}\)

This downloads all genomes on ncbi for a given category looking at their ftp. This could be highly redundant.

download_ncbi_refseq_release\(category, \text{email}=\text{'sequana@pasteur.fr'}, \text{outdir}=.'\)

Download all files of type fna from ncbi FTP.

```python
kb = NCBIDownload()
kb.download_ncbi_refseq_release("viral")
```

class NCBITaxonReader\(names=None, nodes=None\)

This class will help in reading, handling, simplifying NCBI taxonomic DB

When downloading NCBI taxonomy DB using e.g. Kraken, we end up with very large files. One is called names.dmp and the other nodes.dmp. They may be introspected or simplified using this class

The names.dmp is just a CSV file. The header looks like:

<table>
<thead>
<tr>
<th>taxid</th>
<th>name</th>
<th>type of name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>all</td>
<td>synonym</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>scientific name</td>
</tr>
<tr>
<td>2</td>
<td>Bacteria</td>
<td>Bacteria \text{&lt;prokaryote&gt;}</td>
</tr>
<tr>
<td>2</td>
<td>Monera</td>
<td>Monera \text{&lt;Bacteria&gt;}</td>
</tr>
<tr>
<td>2</td>
<td>Procaryotae</td>
<td>Procaryotae \text{&lt;Bacteria&gt;}</td>
</tr>
</tbody>
</table>

It is a tabulated file. If we ignore the | signs, it contains 4 columns:

taxid
name
unique name
type of name

The unique name column is generally empty and is dropped internally. There are different types of name, so there can be several rows for a given taxid. For instance for the taxon 1, there is a scientific name and a synonym.

The df_name is a dataframe that stores the taxid, name and type of name in a dataframe.

The second file nodes.dmp looks like:

<table>
<thead>
<tr>
<th>taxid</th>
<th>parent taxid</th>
<th>rank</th>
<th>type</th>
<th>type</th>
<th>type</th>
<th>type</th>
<th>type</th>
<th>type</th>
<th>type</th>
<th>type</th>
<th>type</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>131567</td>
<td>superkingdom</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>335928</td>
<td>genus</td>
<td>0</td>
<td>1</td>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>species</td>
<td>AC</td>
<td>0</td>
<td>1</td>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>32199</td>
<td>species</td>
<td>BA</td>
<td>0</td>
<td>1</td>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Again this is a tabulated file. The first three columns are taxid, parent taxid, and rank. Rank is species, genus, family, phylum, etc. Newest version of nodes.dmp has only 4 columns (taxid, parent taxid, rank, a dash)

```
from sequana.databases import NCBITaxonReader

# The first time you may want to download the taxdump files
n = NCBITaxonReader()
n.download_taxdump()
n.init("names.dmp", "nodes.dmp")

# next time, you can read it directly
n.NCBITaxonReader("names.dmp", "nodes.dmp")
```
**Constructor**

Parameters

- `names (str)` -- Defaults to "names.dmp".
- `nodes (str)` -- Defaults to "nodes.dmp".

`download_taxdump(outpath='')`

`filter_names_dmp_file(output='names_filtered.dmp', taxons=[])`

Save a subset of nodes.dmp given list of valid taxons

Parameters

- `str` -- Defaults to "nodes_filtered.dmp".
- `taxons (list)` --

`filter_nodes_dmp_file(output='nodes_filtered.dmp', taxons=[])`

Save a subset of nodes.dmp given list of valid taxons

Parameters

- `str` -- Defaults to "nodes_filtered.dmp".
- `taxons (list)` --

`ftp_url = 'ftp.ncbi.nih.gov'`

`get_average_name_per_taxon()`

Return number of rows/names per node/taxon

`get_family(taxon)`

Get all parent taxons

`get_number_taxon()`

Return number of unique taxon

`get_scientific_name(taxon)`

Return scientific name of a given Taxon

`get_taxon_from_scientific_name(scname)`

Return taxon corresponding to a scientific name

return: unique taxon or first one found. If none found, returns None

`init(names, nodes)`

`search(name)`

Search names column
2.12.7 Enrichment

2.12.8 Experimental design

IEM class

class IEM(
    filename, tryme=False
)

Reader and validator of IEM samplesheets

Sections are case-sensitive and denoted by a line starting and ending with square brackets. Except for commas
and end of line, no extra character after the ending square bracket are authorised.

Sample sheet must begin with the [Header] section and end with the [Data] section. Others can be ordered arbitraly.

**Header** section must be on the first line. It contains records represented as a series of key-value pairs. So, each
line requires exactly two fields.

**Settings** is an optional section with key-value pairs.

**Reads** contains number of cycles per read. Only required for MiSeq.

For adapters IEM sample sheet those fields are known to be present: [Version], [Name], [settings], [I7], [I5],
[IndexPlateLayout].

**Data** section: it is required and must be located at the end of the Sample Sheet file. The Data section is a CSV-like
table.

No specific ordering of the column names is required and they are not case-sensitive. At a minimum, the one
column that is universally required is Sample_ID, which provides a unique string identifier for each sample.

Example of typical Data section to be used with bcl2fastq:

```
[Header]

[Data]
Sample_ID,Sample_Name,I7_Index_ID,index,I5_Index_ID,index2
A10001,Sample_A,D701,AATACTCG,D501,TATAGCCT
A10002,Sample_B,D702,TCCGGAGA,D501,TATAGCCT
A10003,Sample_C,D703,CGCTCATT,D501,TATAGCCT
A10004,Sample_D,D704,GAGATTCC,D501,TATAGCCT
```

**References** illumina specifications 970-2017-004.pdf

**property df**

**property header**

**property index_adapters**

**property instrument**

**property name**

**quick_fix**(output_filename)

**property samples**

**property settings**

2.12. References
**to_fasta**(adapter_name="")

**validate()**

This method checks whether the sample sheet is correctly formatted

**Checks for:**

- presence of ; at the end of lines indicated an edition with excel that wrongly transformed the data into a pure CSV file
- inconsistent numbers of columns in the [DATA] section, which must be CSV-like section
- Extra lines at the end are ignored
- special characters except are forbidden except - and _

**property version**

### 2.12.9 FASTQ module

Utilities to manipulate FASTQ and Reads

**class FastQ**(filename, verbose=False)

Class to handle FastQ files

Some of the methods are based on pysam but a few are also original to sequana. In general, input can be zipped or not and output can be zipped or not (based on the extension).

An example is the `extract_head()` method:

```python
f = FastQ("input_file.fastq.gz")
f.extract_head(100000, output='test.fastq')
f.extract_head(100000, output='test.fastq.gz')
```

equivalent to:

```bash
zcat myreads.fastq.gz | head -100000 | gzip > test100k.fastq.gz
```

An efficient implementation to count the number of lines is also available:

```python
f.count_lines()
```

or reads (assuming 4 lines per read):

```python
f.count_reads()
```

Operators available:

- equality ==

**count_lines()**

Return number of lines

**count_reads()**

Return count_lines divided by 4
extract_head(N, output_filename)
Extract the heads of a FastQ files

Parameters

- N (int) --
- output_filename (str) -- Based on the extension the output file is zipped or not (.gz extension only)

This function is convenient since it takes into account the input file being compressed or not and the output file being compressed or not. It is in general 2-3 times faster than the equivalent unix commands combined together but is 10 times slower for the case on uncompressed input and uncompressed output.

**Warning:** this function extract the N first lines and does not check if there are empty lines in your FastQ/FastA files.

filter(identifiers_list=[], min_bp=None, max_bp=None, progressbar=True, output_filename='filtered.fastq')
Save reads in a new file if there are not in the identifier_list

Parameters

- min_bp (int) -- ignore reads with length shorter than min_bp
- max_bp (int) -- ignore reads with length above max_bp

get_lengths()

joining(pattern, output_filename)
not implemented

zcat Block*.fastq.gz | gzip > combined.fastq.gz

property n_lines
return number of lines (should be 4 times number of reads)

property n_reads
return number of reads

next()

rewind()
Allows to iter from the beginning without openning the file or creating a new instance.

select_random_reads(N=None, output_filename='random.fastq')
Select random reads and save in a file

Parameters

- N (int) -- number of random unique reads to select should provide a number but a list can be used as well. You can select random reads for R1, and re-use the returned list as input for the R2 (since pairs must be kept)
- output_filename (str) --

If you have a pair of files, the same reads must be selected in R1 and R2.
```python
f1 = FastQ(file1)
selection = f1.select_random_reads(N=1000)
f2 = FastQ(file2)
f2.select_random_reads(selection)
```

Changed in version 0.9.8: use list instead of set to keep integrity of paired-data

```python
def select_reads(read_identifiers, output_filename=None, progress=True)
    identifiers must be the name of the read without starting @ sign and without comments.
```

```python
def split_chunks(N=10)
    Not implemented
```

```python
def split_lines(N=100000, gzip=True)
    Not implemented
```

```python
def stats()
```

```python
def to_fasta(output_filename='test.fasta')
    Slow but works for now in pure python with input compressed data.
```

```python
def to_kmer_content(k=7)
    Return a Series with kmer count across all reads
    Parameters k (int) -- (default to 7-mers)
    Returns Pandas Series with index as kmer and values as count.
    Takes about 30 seconds on a million reads.
```

```python
def to_krona(k=7, output_filename='fastq.krona')
    Save Krona file with ACGT content within all k-mers
    Parameters k (int) -- (default to 7-mers)
    Save results in file, which can then be translated into a HTML file using:
    ```
    ktImportText fastq.krona
    open text.krona.html
    ```
```

```
class FastQC(filename, max_sample=500000, verbose=True, skip_nrows=0)
    Simple QC diagnostic
    Similarly to some of the plots of FastQC tools, we scan the FastQ and generates some diagnostic plots. The interest is that we'll be able to create more advanced plots later on.
    Here is an example of the boxplot quality across all bases:
```

```python
from sequana import sequana_data
from sequana import FastQC
filename = sequana_data("test.fastq", "doc")
qc = FastQC(filename)
qc.boxplot_quality()
```

**Warning:** some plots will work for Illumina reads only right now
Note: Although all reads are parsed (e.g. to count the number of nucleotides, some information uses a limited number of reads (e.g. qualities), which is set to 500,000 by deafult.

constructor

Parameters

- `filename` --
- `max_sample (int)` -- Large files will not fit in memory. We therefore restrict the numbers of reads to be used for some of the statistics to 500,000. This also reduces the amount of time required to get a good feeling of the data quality. The entire input file is parsed tough. This is required for instance to get the number of nucleotides.

`boxplot_quality(hold=False, ax=None)`

Boxplot quality

 Same plots as in FastQC that is average quality for all bases. In addition a 1 sigma error envelope is shown (yellow).

 Background separate zone of good, average and bad quality (arbitrary).

`get_actg_content()`

`get_stats()`

`histogram_gc_content()`

Plot histogram of GC content

```python
from sequana import sequana_data
from sequana import FastQC
filename = sequana_data("test.fastq", "doc")
qc = FastQC(filename)
qc.histogram_gc_content()
```

`histogram_sequence_lengths(logy=True)`

Histogram sequence lengths

```python
from sequana import sequana_data
from sequana import FastQC
filename = sequana_data("test.fastq", "doc")
qc = FastQC(filename)
qc.histogram_sequence_lengths()
```

`plot_acgt_content(stacked=False)`

Plot histogram of GC content

```python
from sequana import sequana_data
from sequana import FastQC
filename = sequana_data("test.fastq", "doc")
qc = FastQC(filename)
qc.plot_acgt_content()
```
class Identifier(identifier, version='unknown')
Class to interpret Read's identifier

Warning: Implemented for Illumina 1.8+ and 1.4. Other cases will simply stored the identifier without
interpretation

```python
>>> from sequana import Identifier
>>> ident.info['x_coordinate']
2
```

Currently, the following identifiers will be recognised automatically:

**Illumina_1.4** An example is

```
@HWUSI-EAS100R:6:73:941:1973#0/1
```

**Illumina_1.8** An example is:

```
@EAS139:136:FC706VJ:2:2104:15343:197393 1:Y:18:ATCACG
```

Other that could be implemented are NCBI

```
@FSRRS4401BE7HA [length=395] [gc=36.46] [flows=800] [phred_min=0]
[phred_max=40] [trimmed_length=95]
```

Information can also be found here http://support.illumina.com/help/SequencingAnalysisWorkflow/Content/
Vault/Informatics/Sequencing_Analysis/CASAVA/swSEQ_mCA_FASTQFiles.htm

is_fastq(filename)

### 2.12.10 FASTA module

Utilities to manipulate FastA files

class FastA(filename, verbose=False)
Class to handle FastA files

```python
from sequana import FastA
f = FastA("test.fa")
read = next(f)
names = f.names
```

**GC_content()**
Return GC content in percentage of all sequences found in the FastA file

**GC_content_sequence(sequence)**
Return GC content in percentage of a sequence

property comments
**explode**(`outdir='.'

extract sequences from original file and save them into individual files

**filter**(`output_filename`, `names_to_keep=None`, `names_to_exclude=None`)

save FastA excluding or including specific sequences

**format_contigs_denovo**(`output_file`, `len_min=500`)

Remove contigs with sequence length below specific threshold.

**Parameters**

- `output_file` (**str**) -- output file name.
- `len_min` (**int**) -- minimal length of contigs.

**Example:**

```python
from sequana import FastA
contigs = FastA("denovo_assembly.fasta")
contigs.format_contigs_denovo("output.fasta", len_min=500)
```

**get_lengths_as_dict**()

Return dictionary with sequence names and lengths as keys/values

**get_stats**()

Return a dictionary with basic statistics

**property lengths**

**property names**

**next**()

**reverse_and_save**(`filename`)

Reverse sequences and save in a file

**save_collapsed_fasta**(`outfile`, `ctgname`, `width=80`, `comment=None`)

Concatenate all contigs and save results

**save_ctg_to_fasta**(`ctgname`, `outname`, `max_length=-1`)

Select a contig and save in a file

**select_random_reads**(`N=None`, `output_filename='random.fasta'`)

Select random reads and save in a file

**Parameters**

- `N` (**int**) -- number of random unique reads to select should provide a number but a list can be used as well.
- `output_filename` (**str**) --

**property sequences**

**summary**(`max_contigs=-1`)

returns summary and print information on the stdout

This method is used when calling sequana standalone
Sequana, Release 0.14.0

```python
sequana summary test.fasta

to fasta (outfile, width=80)
Save the input FastA file into a new file
The interest of this method is to wrap the sequence into 80 characters. This is useful if the input file is not
formatted correctly.

to igv chrom size (output)
Create a IGV file storing chromosomes and their sizes

2.12.11 Feature counts module

feature counts related tools

```python
class FeatureCount (filename, clean_sample_names=True, extra_name_rm=['Aligned'], drop_loc=True,
guess_design=False)

Read a featureCounts output file.
The input file is expected to be generated with featurecounts tool. It should be a TSV file such as the following
one with the header provided herebelow. Of course input BAM files can be named after your samples:

<table>
<thead>
<tr>
<th>Geneid</th>
<th>Chr</th>
<th>Start</th>
<th>End</th>
<th>Strand</th>
<th>Length</th>
<th>WT1</th>
<th>WT2</th>
<th>WT3</th>
<th>KO1</th>
<th>KO2</th>
<th>KO3</th>
</tr>
</thead>
<tbody>
<tr>
<td>gene1</td>
<td>NC_010602.1</td>
<td>141</td>
<td>1466</td>
<td>+</td>
<td>1326</td>
<td>11</td>
<td>20</td>
<td>15</td>
<td>13</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>gene2</td>
<td>NC_010602.1</td>
<td>1713</td>
<td>2831</td>
<td>+</td>
<td>1119</td>
<td>35</td>
<td>54</td>
<td>58</td>
<td>34</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>gene3</td>
<td>NC_010602.1</td>
<td>2831</td>
<td>3934</td>
<td>+</td>
<td>1104</td>
<td>9</td>
<td>16</td>
<td>16</td>
<td>4</td>
<td>18</td>
<td>18</td>
</tr>
</tbody>
</table>

```python
from sequana import FeatureCount
fc = FeatureCount("all_features.out", extra_name_rm="_S\d+")
fc.rnadiff_df.to_csv("fc.csv")

Constructor

Get the featureCounts output as a pandas DataFrame

Parameters

- **clean_sample_names** *(bool)* -- if simplifying the sample names in featureCount output columns
- **extra_name_rm** *(list)* -- extra list of regex to remove from samples_names (ignored if
  clean sample name is False)
- **drop_loc** *(bool)* -- if dropping the extra location columns (ie getting only the count matrix)

property df

```python
class FeatureCountMerger (pattern="*feature.out", fof=[])

Merge several feature counts files

to tsv (output_filename="all_features.out")

```

112 Chapter 2. User guide and reference
class MultiFeatureCount(rnaseq_folder='.', tolerance=0.1)

Read set of feature counts using different options of strandness

```python
from sequana import sequana_data
from sequana.featurecounts import *
directory = sequana_data("/rnaseq_0")
ff = MultiFeatureCount(directory, 0.15)
ff._get_most_probable_strand_consensus()
ff.plot_strandness()
```

See also:

`get_most_probable_strand()` for more information about the tolerance parameter and meaning of strandness.

The expected data structure is

```
rnaseq_folder
  ├── sample1
  │    ├── feature_counts_0
  │    │    └── sample_feature.out
  │    ├── feature_counts_1
  │    │    └── sample_feature.out
  │    └── feature_counts_2
  │        └── sample_feature.out
  └── sample2
      ├── feature_counts_0
      │    └── sample_feature.out
      ├── feature_counts_1
      │    └── sample_feature.out
      └── feature_counts_2
          └── sample_feature.out
```

The new expected data structure is

```
new_rnaseq_output/
  ├── sample1
  │    └── feature_counts
  │        └── 0
  │            └── sample_feature.out
  │        └── 1
  │            └── sample_feature.out
  │        └── 2
  │            └── sample_feature.out
  └── sample2
      └── feature_counts
          └── 0
              └── sample_feature.out
          └── 1
              └── sample_feature.out
          └── 2
              └── sample_feature.out
```

Parameters
• `rnaseq_folder` *(str)* --
• `tolerance` *(int)* -- the tolerance between 0 and 0.25

**plot_strandness** *(fontsize=12, output_filename='strand_summary.png', savefig=False)*

**get_most_probable_strand** *(filenames, tolerance, sample_name)*

Return most probable strand given 3 feature count files (strand of 0, 1, and 2)

Return the total counts by strand from featureCount matrix folder, strandness and probable strand for a single sample (using a tolerance threshold for strandness). This assumes a single sample by featureCounts file.

**Parameters**

• `filenames` -- a list of 3 feature counts files for a given sample corresponding to the strand 0, 1, 2
• `tolerance` -- a value below 0.5
• `sample` -- the name of the sample corresponding to the list in filenames

Possible values returned are:

• 0: unstranded
• 1: stranded
• 2: eversely stranded

We compute the number of counts in case 1 and 2 and compute the ratio strand as $RS = \text{stranded}/(\text{stranded} + \text{reverselystranded})$. Then we decide on the possible strandness with the following criteria:

• if $RS < \text{tolerance}$, reversedly stranded
• if $RS$ in $0.5+-\text{tolerance}$: unstranded.
• if $RS > 1-\text{tolerance}$, stranded
• otherwise, we cannot decided.

**get_most_probable_strand_consensus** *(rnaseq_folder, tolerance, sample_pattern='*/feature_counts_[012]', file_pattern='feature_counts_[012]/*_feature.out',)*

From a sequana RNA-seq run folder get the most probable strand, based on the frequencies of counts assigned with '0', '1' or '2' type strandness (featureCounts nomenclature) across all samples.

**Parameters**

• `rnaseq_folder` -- the main directory
• `tolerance` -- a value in the range 0-0.5. typically 0.1 or 0.15
• `pattern` -- the samples directory pattern
• `pattern_file` -- the feature counts pattern

If guess is not possible given the tolerance, fills with None

Consider this tree structure:

```
rnaseq_folder
  |-- sample1
  |    |-- feature_counts
  |    |    |-- 0
  |    |    |    |-- sample_feature.out
```

(continues on next page)
Then, the following command should all files and report the most probable strand (0,1,2) given the sample1 and sample2:

```python
get_most_probable_strand_consensus("rnaseq_folder", 0.15)
```

This tree structure is understood automatically. If you have a different one, you can set the pattern (for samples) and pattern_files parameters.

### 2.12.12 Sequence module

**class DNA**

```python
class DNA(sequence, codons_stop=['TAA', 'TGA', 'TAG'], codons_stop_rev=['TTA', 'TCA', 'CTA'],
codons_start=['ATG'], codons_start_rev=['CAT'])
```

Simple DNA class

```python
>>> from sequana.sequence import DNA
>>> d = DNA("ACGTTTT")
>>> d.reverse_complement()
```

Some long computations are done when setting the window size:

```python
d.window = 100
```

The ORF detection has been validated agains a plasmodium 3D7 ORF file found on plasmodb.org across the 14 chromosomes.

**Constructor**

A sequence is just a string stored in the `sequence` attribute. It has properties related to the type of alphabet authorised.

**Parameters**

- `sequence (str)` -- May be a string of a Fasta File, in which case only the first sequence is used.
- `complement_in
- `complement_out
- `letters` -- authorise letters. Used in check() only.

2.12. References 115
Todo: use counter only once as a property

property AT_skew
property GC_skew
property ORF_pos
barplot_count_ORF_CDS_by_frame(alpha=0.5, bins=40, xlabel='Frame', ylabel='#', bar_width=0.35)
entropy(sequence)
hist_ORF_CDS_linescale(alpha=0.5, bins=40, xlabel='Length', ylabel='#')
hist_ORF_CDS_logscale(alpha=0.5, bins=40, xlabel='Length', ylabel='#')
plot_all_skews(figsize=(10, 12), fontsize=16, alpha=0.5)
property threshold
property type_filter
property type_window
property window

class RNA(sequence)
Simple RNA class

>>> from sequana.sequence import RNA
>>> d = RNA("ACGUUUU")
>>> d.reverse_complement()

Constructor

A sequence is just a string stored in the sequence attribute. It has properties related to the type of alphabet authorised.

Parameters

- **sequence** (str) -- May be a string of a Fasta File, in which case only the first sequence is used.
- **complement_in** --
- **complement_out** --
- **letters** -- authorise letters. Used in check() only.

Todo: use counter only once as a property

class Repeats(filename_fasta, merge=False, name=None)
Class for finding repeats in DNA or RNA linear sequences.

Computation is performed each time the threshold is set to a new value.
```python
from sequana import sequana_data, Repeats
rr = Repeats(sequana_data("measles.fa"))
rr.threshold = 4
rr.hist_length_repeats()
```

**Note:** Works with shustring package from Bioconda (April 2017)

**Todo:** use a specific sequence (first one by default). Others can be selected by name

### Constructor

Input must be a fasta file with valid DNA or RNA characters

**Parameters**

- `filename_fasta (str)` -- a Fasta file, only the first sequence is used !
- `threshold (int)` -- Minimal length of repeat to output
- `name (str)` -- if name is provided, scan the Fasta file and select the corresponding sequence.
  if you want to analyse all sequences, you need to use a loop by setting _header for each
  sequence with the sequence name found in sequence header.

**Note:** known problems. Header with a > character (e.g. in the comment) are left strip and only the comments
is kept. Another issue is for multi-fasta where one sequence is ignored (last or first ?)

### property begin_end_repeat_position

### property df_shustring

Return dataframe with shortest unique substring length at each position shortest unique substrings are
unique in the sequence and its complement Uses shustring tool

### property do_merge

### property header

get first line of fasta (needed in input shustring) and replace spaces by underscores

```
hist_length_repeats(bins=20, alpha=0.5, hold=False, fontsize=12, grid=True, title='Repeat length',
xlabel='Repeat length', ylabel='#', logy=True)
```

Plots histogram of the repeat lengths

### property length

### property list_len_repeats

### property longest_shustring

### property names

### plot(clf=True)

### property threshold

---

2.12. References
class Sequence(sequence, complement_in=b'ACGT', complement_out=b'TGCA', letters='ACGT')

Abstract base clase for other specialised sequences such as DNA.

Sequenced is the base class for other classes such as DNA and RNA.

from sequana import Sequence
s = Sequence("ACGT")
s.stats()
s.get_complement()

Note: You may use a Fasta file as input (see constructor)

Constructor

A sequence is just a string stored in the sequence attribute. It has properties related to the type of alphabet authorised.

Parameters

- sequence (str) -- May be a string of a Fasta File, in which case only the first sequence is used.
- complement_in --
- complement_out --
- letters -- authorise letters. Used in check() only.

Todo: use counter only once as a property

check()

Check that all letters are valid

complement()

Alias to get_complement()

gc_content()

Return mean GC content

gc_content()()

Return complement

gc_content()()

Return complement

get_occurrences(pattern, overlap=False)

Return position of the input pattern in the sequence

>>> from sequana import Sequence
>>> s = Sequence('ACGTTTTACGT')
>>> s.get_occurrences('ACGT')
[0, 7]

gc_content()()

Return reverse sequence
get_reverse_complement()
    Return reverse complement

reverse()
    Alias to get_reverse()

reverse_complement()
    Alias to get_reverse_complement

property sequence

stats()
    Return basic stats about the number of letters

2.12.13 Kmer module

build_kmer(length=6, letters='CG')
    Return list of kmer of given length based on a set of letters
    Returns list of kmers

get_kmer(sequence, k=7)
    Given a sequence, return consecutive kmers
    Returns iterator of kmers

2.12.14 Taxonomy related (Kraken - Krona)

class KrakenAnalysis(fastq, database, threads=4, confidence=0)
    Run kraken on a set of FastQ files
    In order to run a Kraken analysis, we first need a local database. We provide a Toy example. The ToyDB is downloadable as follows (you will need to run the following code only once):

```
from sequana import KrakenDownload
kd = KrakenDownload()
kd.download_kraken_toydb()
```

See also:

KrakenDownload for more databases

The path to the database is required to run the analysis. It has been stored in the directory $config/sequana/kraken_toydb under Linux platforms The following code should be platform independent:

```
import os
from sequana import sequana_config_path
database = sequana_config_path + os.sep + "kraken_toydb"
```

Finally, we can run the analysis on the toy data set:

```
from sequana import sequana_data
data = sequana_data("Hm2_GTGAAA_L005_R1_001.fastq.gz", "data")
ka = KrakenAnalysis(data, database=database)
ka.run()
```

2.12. References
This creates a file named *kraken.out*. It can be interpreted with *KrakenResults*

**Constructor**

Parameters

- **fastq** -- either a fastq filename or a list of 2 fastq filenames
- **database** -- the path to a valid Kraken database
- **threads** -- number of threads to be used by Kraken
- **confidence** -- parameter used by kraken2
- **return** --

```python
run(output_filename=None, output_filename_classified=None, output_filename_unclassified=None, only_classified_output=False)
```

Performs the kraken analysis

Parameters

- **output_filename** (str) -- if not provided, a temporary file is used and stored in `kraken_output`
- **output_filename_classified** (str) -- not compressed
- **output_filename_unclassified** (str) -- not compressed

**class** *KrakenDB*(filename)

Class to handle a kraken DB

**property** *version*

**class** *KrakenDownload*(output_dir=None)

Utility to download Kraken DB and place them in a local directory

```python
from sequana import KrakenDownload
kd = KrakenDownload()
k.d.download('toydb')
```

**class** *KrakenPipeline*(fastq, database, threads=4, output_directory='kraken', dbname=None, confidence=0)

Used by the standalone application sequana_taxonomy

This runs Kraken on a set of FastQ files, transform the results in a format compatible for Krona, and creates a Krona HTML report.

```python
from sequana import KrakenPipeline
kt = KrakenPipeline(\['R1.fastq.gz', 'R2.fastq.gz'\], database='krakendb')
kt.run()
kt.show()
```

**Warning:** We do not provide Kraken database within sequana. You may either download a database from [https://ccb.jhu.edu/software/kraken/](https://ccb.jhu.edu/software/kraken/) or use this class to download a toy example that will be stored in e.g. .config/sequana under Unix platforms. See *KrakenDownload*.
See also:

We provide a standalone application of this class, which is called sequana_taxonomy and can be used within a command shell.

Constructor

Parameters

- **fastq** -- either a fastq filename or a list of 2 fastq filenames
- **database** -- the path to a valid Kraken database
- **threads** -- number of threads to be used by Kraken
- **output_directory** -- output filename of the Krona HTML page
- **dbname** --

Description: internally, once Kraken has performed an analysis, reads are associated to a taxon (or not). We then find the corresponding lineage and scientific names to be stored within a Krona formatted file. KtImportTex is then used to create the Krona page.

```python
run(output_filename_classified=None, output_filename_unclassified=None, only_classified_output=False)
```

Run the analysis using Kraken and create the Krona output

**Todo:** reuse the KrakenResults code to simplify this method.

```python
show()
```

Opens the filename defined in the constructor

**class KrakenResults(filename='kraken.out', verbose=True)**

Translate Kraken results into a Krona-compatible file

If you run a kraken analysis with `KrakenAnalysis`, you will end up with a file e.g. named kraken.out (by default).

You could use kraken-translate but then you need extra parsing to convert into a Krona-compatible file. Here, we take the output from kraken and directly transform it to a krona-compatible file.

kraken2 uses the --use-names that needs extra parsing.

```python
k = KrakenResults("kraken.out")
k.kraken_to_krona()
```

Then format expected looks like:

```
```

Where each row corresponds to one read.

"562:13 561:4 A:31 0:1 562:3" would indicate that:

- the first 13 k-mers mapped to taxonomy ID #562
- the next 4 k-mers mapped to taxonomy ID #561
- the next 31 k-mers contained an ambiguous nucleotide

(continues on next page)
the next k-mer was not in the database
the last 3 k-mers mapped to taxonomy ID #562

For kraken2, format is slightly different since it depends on paired or not. If paired,

C read1 2697049 151|151 2697049:117 |:| 0:1 2697049:116

See kraken documentation for details.

Note: a taxon of ID 1 (root) means that the read is classified but in a different domain. https://github.com/DerrickWood/kraken/issues/100

Note: This takes care of fetching taxons and the corresponding lineages from online web services.

**constructor**

Parameters `filename` -- the input from KrakenAnalysis class

**boxplot_classified_vs_read_length()**

Show distribution of the read length grouped by classified or not

**property df**

**get_taxonomy_db(ids)**

Retrieve taxons given a list of taxons

Parameters `ids (list)` -- list of taxons as strings or integers. Could also be a single string or a single integer

Returns a dataframe

Note: the first call first loads all taxons in memory and takes a few seconds but subsequent calls are much faster

**histo_classified_vs_read_length()**

Show distribution of the read length grouped by classified or not

**kraken_to_csv(filename, dbname)**

**kraken_to_json(filename, dbname)**

**kraken_to_krona(output_filename=None, nofile=False)**

Returns status: True is everything went fine otherwise False

**plot(kind='pie', cmap='tab20c', threshold=1, radius=0.9, textcolor='red', delete_krona_file=False, **kargs)**

A simple non-interactive plot of taxons

Returns None if no taxon were found and a dataframe otherwise

A Krona Javascript output is also available in **kraken_to_krona()**
```python
from sequana import KrakenResults, sequana_data
test_file = sequana_data("kraken.out", "doc")
k = KrakenResults(test_file)
df = k.plot(kind='pie')
```

See also:
to generate the data see KrakenPipeline or the standalone application sequana_taxonomy.

**Todo:** For a future release, we could use this kind of plot https://stackoverflow.com/questions/57720935/
how-to-use-correct-cmap-colors-in-nested-pie-chart-in-matplotlib

```python
plot2(kind='pie', fontsize=12)
```

This is the simplified static krona-like plot included in HTML reports

```python
property taxons
to_js(output='krona.html')
```

class KrakenSequential(filename_fastq, fof_databases, threads=1, output_directory='./kraken_sequential/',
keep_temp_files=False, output_filename_unclassified=None,
output_filename_classified=None, force=False, confidence=0)

Kraken Sequential Analysis

This runs Kraken on a FastQ file with multiple k-mer databases in a sequential way way. Unclassified sequences
with the first database are input for the second, and so on.

The input may be a single FastQ file or paired, gzipped or not. FastA are also accepted.

**constructor**

**Parameters**

- **filename_fastq** -- FastQ file to analyse
- **fof_databases** -- file that contains a list of databases paths (one per line). The order is important. Note that you may also provide a list of database paths.
- **threads** -- number of threads to be used by Kraken
- **output_directory** -- name of the output directory
- **keep_temp_files** -- bool, if True, will keep intermediate files from each Kraken analysis, and save html report at each step
- **force** (bool) -- if the output directory already exists, the instanciation fails so that the existing data is not overwritten. If you wish to overwrite the existing directory, set this parameter to True.

```python
run(dbname='multiple', output_prefix='kraken_final')
```

Run the sequential analysis

**Parameters**

- **dbname** --
- **output_prefix** --
**Returns** dictionary summarizing the databases names and classified/unclassified

This method does not return anything creates a set of files:

- kraken_final.out
- krona_final.html
- kraken.png (pie plot of the classified/unclassified reads)

**Note:** the databases are run in the order provided in the constructor.

```python
class MultiKrakenResults(filenames, sample_names=None)
```

Select several kraken output and creates summary plots

```python
import glob
mkr = MultiKrakenResults(glob.glob("*/kraken.csv"))
mkr.plot_stacked_hist()
```

```python
def get_df(limit=5)

def plot_stacked_hist(output_filename=None, dpi=200, kind='barh', fontsize=10, edgecolor='k', lw=1, width=1, ytick_fontsize=10, max_labels=50, max_sample_name_length=30)
```

Summary plot of reads classified.

```python
class MultiKrakenResults2(filenames, sample_names=None)
```

Select several kraken output and creates summary plots

```python
import glob
mkr = MultiKrakenResults2(glob.glob("/*/summary.json"))
mkr.plot_stacked_hist()
```

```python
def get_df(limit=5, sorting_method='sample_name')

def plot_stacked_hist(output_filename=None, dpi=200, kind='barh', fontsize=10, edgecolor='k', lw=2, width=1, ytick_fontsize=10, max_labels=50, max_sample_name_length=30)
```

Summary plot of reads classified.

**Parameters**

- **sorting_method** -- only by sample name for now
- **cmap** -- a valid matplotlib colormap. viridis is the default sequana colormap.

If you prefer to use a colormap, you can use:

```python
from matplotlib import cm
cm = matplotlib.colormap
colors = [cm.get_cmap(cmap)(x) for x in pylab.linspace(0.2, 1, L)]
```

```python
class NCBIHierarchy(names, nodes)
```

**Parameters**

- **names** -- can be a local file or URL
- **nodes** -- can be a local file or URL
create_taxonomy_file(filename='taxonomy.dat')

class Taxonomy(*args, **kwargs)
    This class should ease the retrieval and manipulation of Taxons

    There are many resources to retrieve information about a Taxon. For instance, from BioServices, one can use UniProt, Ensembl, or EUtils. This is convenient to retrieve a Taxon (see fetch_by_name() and fetch_by_id() that rely on Ensembl). However, you can also download a flat file from EBI ftp server, which stores a set of records (2.8M (april 2020).

    Note that the Ensembl database does not seem to be as up to date as the flat files but entries contain more information.

    For instance taxon 2 is in the flat file but not available through the fetch_by_id(), which uses ensembl.

    So, you may access to a taxon in 2 different ways getting different dictionary. However, 3 keys are common (id, parent, scientific_name)

    >>> t = taxonomy.Taxonomy()
    >>> t.fetch_by_id(9606) # Get a dictionary from Ensembl
    >>> t.records[9606] # or just try with the get
    >>> t[9606]
    >>> t.get_lineage(9606)

    Possible ranks are various. You may have biotype, clade, etc but generally speaking ranks are about lineage. For a given rank, e.g. kingdom, you may have sub division such as superkingdom and subkingdom. order has even more subdivisions (infra, parv, sub, super)

    constructor

    Parameters
    
    • offline -- if you do not have internet, the connection to Ensembl may hang for a while and fail. If so, set offline to True
    
    • from -- download taxonomy databases from ncbi

    append_existing_database(filename)
    Taxonomy DB looks like:

    ID : 2731450
    PARENT ID : 1914233
    RANK : genus
    SCIENTIFIC NAME : Limnoglobus

    #

    a = NCBITaxonomy("names.dmp", "nodes.dmp")
a.create_taxonomy_file("taxonomy.dat")
tax = Taxonomy()
tax.append_existing_database("taxonomy.dat")

download_taxonomic_file(overwrite=False)
    Loads entire flat file from EBI
    Do not overwrite the file by default.

2.12. References
**fetch_by_id**(taxon)

Search for a taxon by identifier

:return: a dictionary.

```python
>>> ret = s.search_by_id('10090')
>>> ret['name']
'Mus Musculus'
```

**fetch_by_name**(name)

Search a taxon by its name.

Parameters

name (str) -- name of an organism. SQL cards possible e.g., _ and % characters.

Returns a list of possible matches. Each item being a dictionary.

```python
>>> ret = s.search_by_name('Mus Musculus')
>>> ret[0]['id']
10090
```

**find_taxon**(taxid, mode='ncbi')

**get_children**(taxon)

**get_lineage**(taxon)

Get lineage of a taxon

Parameters
taxon (int) -- a known taxon

Returns list containing the lineage

**get_lineage_and_rank**(taxon)

Get lineage and rank of a taxon

Parameters
taxon (int) --

Returns a list of tuples. Each tuple is a pair of taxon name/rank The list is the lineage for to the input taxon.

**get_names_for_given_rank**(rank)

**get_parent_name**(taxon)

**get_parent_taxon**(taxon)

**get_ranks**()

**get_record_for_given_rank**(rank)

**load_records**(overwrite=False)

Load a flat file and store records in records

Since version 0.8.3 we use NCBI that is updated more often than the ebi ftp according to their README.

2.12.15 Pacbio module

Pacbio QC and stats

class BAMSimul(filename)
    
    BAM reader for Pacbio simulated reads (PBsim)

    A summary of the data is stored in the attribute df. It contains information such as the length of the reads, the ACGT content, the GC content.

    Constructor

    Parameters filename (str) -- filename of the input pacbio BAM file. The content of the BAM file is not the output of a mapper. Instead, it is the output of a Pacbio (Sequel) sequencing (e.g., subreads).

    property df

    filter_bool(output_filename, mask)

        Select and Write reads using a mask

        Parameters

        • output_filename (str) -- name of output file
        • list_bool (list) -- True to write read to output, False to ignore it

    filter_length(output_filename, threshold_min=0, threshold_max=inf)

        Select and Write reads within a given range

        Parameters

        • output_filename (str) -- name of output file
        • threshold_min (int) -- minimum length of the reads to keep
        • threshold_max (int) -- maximum length of the reads to keep

    hist_GC(bins=50, alpha=0.5, hold=False, fontsize=12, grid=True, xlabel='GC %', ylabel='#', label='', title=None)

        Plot histogram GC content

        Parameters

        • bins (int) -- binning for the histogram
        • alpha (float) -- transparency of the histograms
        • hold (bool) --
        • fontsize (int) -- fontsize of the x and y labels and title.
        • grid (bool) -- add grid or not
        • xlabel (str) --
        • ylabel (str) --
        • label (str) -- label of the histogram (for the legend)
        • title (str) --
```python
from sequana.pacbio import PacbioSubreads
from sequana import sequana_data
b = PacbioSubreads(sequana_data("test_pacbio_subreads.bam"))
b.hist_GC()
```

**hist_read_length**(``bins=50, alpha=0.8, hold=False, fontsize=12, grid=True, xlabel='Read Length', ylabel='#', label='', title=None, logy=False, ec='k', hist_kwargs={})

Plot histogram Read length

**Parameters**

- **bins** (`int`) -- binning for the histogram
- **alpha** (`float`) -- transparency of the histograms
- **hold** (`bool`) --
- **fontsize** (`int`) --
- **grid** (`bool`) --
- **xlabel** (`str`) --
- **ylabel** (`str`) --
- **label** (`str`) -- label of the histogram (for the legend)
- **title** (`str`) --

```python
from sequana.pacbio import PacbioSubreads
from sequana import sequana_data
b = PacbioSubreads(sequana_data("test_pacbio_subreads.bam"))
b.hist_read_length()
```

**plot_GC_read_len**(``hold=False, fontsize=12, bins=[200, 60], grid=True, xlabel='GC %', ylabel='#', cmap='BrBG'``)

Plot GC content versus read length

**Parameters**

- **hold** (`bool`) --
- **fontsize** (`int`) -- for x and y labels and title
- **bins** -- a integer or tuple of 2 integers to specify the binning of the x and y 2D histogram.
- **grid** (`bool`) --
- **xlabel** (`str`) --
- **ylabel** (`str`) --

```python
from sequana.pacbio import PacbioSubreads
from sequana import sequana_data
b = PacbioSubreads(sequana_data("test_pacbio_subreads.bam"))
b.plot_GC_read_len(bins=[10, 10])
```

**reset()**
to_fasta(output_filename, threads=2)
    Export BAM reads into a Fasta file

Parameters

• output_filename -- name of the output file (use .fasta extension)

• threads (int) -- number of threads to use

Note: this executes a shell command based on samtools

Warning: this takes a few minutes for 500,000 reads

to_fastq(output_filename, threads=2)
    Export BAM reads into FastQ file

class Barcoding(filename)
    Read as input a file created by smrtlink that stores statistics about each barcode. This is a simple CSV file with one line per barcode

    hist_mean_polymerase_read_length(bins=10, fontsize=12)

    hist_polymerase_per_barcode(bins=10, fontsize=12)
        histogram of number of polymerase per barcode
        Cumulative histogram gives total number of polymerase reads

    hist_quality_per_barcode(bins=10, fontsize=12)

    plot_and_save_all(dpi=100, directory='.'

    plot_polymerase_per_barcode(fontsize=12, unbarcoded=True)
        Number Of Polymerase Reads Per Barcode

    plot_subreads_histogram(bins=10, fontsize=12)

class PBSim(input_bam, simul_bam)
    Filter an input BAM (simulated with pbsim) so as to keep reads that fit a target distribution.
    This uses a MH algorithm behind the scene.

    ss = pacbio.PBSim("test10X.bam")
    clf();
    ss.run(bins=100, step=50)

    For example, to simulate data set, use:

    pbsim --data-type CLR --accuracy-min 0.85 --depth 20 --length-mean 8000 --length-sd 800 reference.fasta --model_qc model_qc_clr

    The file model_qc_clr can be retrieved from the github here below.
    See https://github.com/pfaucon/PBSIM-PacBio-Simulator for details.

We get a fastq file where simulated read sequences are randomly sampled from the reference sequence ("reference.fasta") and differences (errors) of the sampled reads are introduced.

The Fastq can be converted to

2.12. References
run\((bins=50, xmin=0, xmax=30000, step=1000, burn=1000, alpha=1, output_filename=None)\)

target_distribution\((xprime)\)

The target distribution

Compute histogram. Get X, Y. Given xprime, interpolate to get yprime use e.g. np.interp

class PacbioMappedBAM\((filename, method)\)

Parameters

filename\((str)\) -- input BAM file

boxplot_mapq_concordance()\)

filter_mapq\((output_filename, threshold_min=0, threshold_max=255)\)

Select and Write reads within a given range

Parameters

• output_filename\((str)\) -- name of output file
• threshold_min\((int)\) -- minimum length of the reads to keep
• threshold_max\((int)\) -- maximum length of the reads to keep

get_coverage\((reference_length=None)\)

hist_GC\((bins=50, alpha=0.5, hold=False, fontsize=12, grid=True, xlabel='GC %', ylabel='#', label='', title=None)\)

Plot histogram GC content

Parameters

• bins\((int)\) -- binning for the histogram
• alpha\((float)\) -- transparency of the histograms
• hold\((bool)\) --
• fontsize\((int)\) -- fontsize of the x and y labels and title.
• grid\((bool)\) -- add grid or not
• xlabel\((str)\) --
• ylabel\((str)\) --
• label\((str)\) -- label of the histogram (for the legend)
• title\((str)\) --

from sequana.pacbio import PacbioSubreads
from sequana import sequana_data
b = PacbioSubreads(sequana_data("test_pacbio_subreads.bam"))
b.hist_GC()

hist_concordance\((bins=100, fontsize=16)\)

formula : 1 - (in + del + mismatch / (in + del + mismatch + match) 

For BWA and BLASR, the get_cigar_stats are different !!! BWA for instance has no X stored while Pacbio forbids the use of the M (CMATCH) tag. Instead, it uses X (CDIFF) and = (CEQUAL) characters.

Subread Accuracy: The post-mapping accuracy of the basecalls. Formula: \([1 - (errors/subread length)]\), where errors = number of deletions + insertions + substitutions.
**hist_median_ccs** *(bins=1000, **kwargs)*

Group subreads by ZMW and plot median of read length for each polymerase

**hist_read_length** *(bins=50, alpha=0.8, hold=False, fontsize=12, grid=True, xlabel='Read Length', ylabel='#', label='', title=None, logy=False, ec='k', hist_kwargs={})*

Plot histogram Read length

---

**Parameters**

- **bins (int)** -- binning for the histogram
- **alpha (float)** -- transparency of the histograms
- **hold (bool)** --
- **fontsize (int)** --
- **grid (bool)** --
- **xlabel (str)** --
- **ylabel (str)** --
- **label (str)** -- label of the histogram (for the legend)
- **title (str)** --

```python
from sequana.pacbio import PacbioSubreads
from sequana import sequana_data
b = PacbioSubreads(sequana_data("test_pacbio_subreads.bam"))
b.hist_read_length()
```

**plot_GC_read_len** *(hold=False, fontsize=12, bins=[200, 60], grid=True, xlabel='GC %', ylabel='#', cmap='BrBG')*

Plot GC content versus read length

---

**Parameters**

- **hold (bool)** --
- **fontsize (int)** -- for x and y labels and title
- **bins** -- a integer or tuple of 2 integers to specify the binning of the x and y 2D histogram.
- **grid (bool)** --
- **xlabel (str)** --
- **ylabel (str)** --

```python
from sequana.pacbio import PacbioSubreads
from sequana import sequana_data
b = PacbioSubreads(sequana_data("test_pacbio_subreads.bam"))
b.plot_GC_read_len(bins=[10, 10])
```

**reset()**

**to_fasta**(output_filename, threads=2)

Export BAM reads into a Fasta file

---

**Parameters**

- **output_filename** -- name of the output file (use .fasta extension)

---

### 2.12. References 131
• **threads** *(int)* -- number of threads to use

**Note:** this executes a shell command based on samtools

**Warning:** this takes a few minutes for 500,000 reads

### to_fastq(output_filename, threads=2)

Export BAM reads into FastQ file

### class PacbioSubreads(filename, sample=0)

BAM reader for Pacbio (reads)

You can read a file as follows:

```python
from sequana.pacbio import Pacbiosubreads
from sequana import sequana_data
filename = sequana_data("test_pacbio_subreads.bam")
b = PacbioSubreads(filename)
```

A summary of the data is stored in the attribute `df`. It contains information such as the length of the reads, the ACGT content, the GC content.

Several plotting methods are available. For instance, `hist_snr()`.

The BAM file used to store the Pacbio reads follows the BAM/SAM specification. Note that the sequence read are termed query, a subsequence of an entire Pacbio ZMW read (a subread), which is basecalls from a single pass of the insert DNA molecule.

In general, only a subsequence of the query will align to the reference genome, and that subsequence is referred to as the aligned query.

When introspecting the aligned BAM file, the extent of the query in ZMW read is denoted as `[qStart, qEnd)` and the extent of the aligned subinterval as `[aStart, aEnd)`. The following graphic illustrates these intervals:

```
<table>
<thead>
<tr>
<th>qStart</th>
<th>qEnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>aStart aEnd</td>
</tr>
<tr>
<td>[---...-----<em>----</em>-------*-----...-----...)</td>
<td>&quot;ZMW read&quot; coord.</td>
</tr>
<tr>
<td>system</td>
<td>~~~~~~~~~~~~~~~~~~</td>
</tr>
<tr>
<td>subseq.</td>
<td>~~~~~~~~~~~~~~~~~~</td>
</tr>
<tr>
<td>coord. system</td>
<td>[---...-----<em>----</em>-------*-----...-----...)</td>
</tr>
<tr>
<td>0</td>
<td>tStart tEnd</td>
</tr>
</tbody>
</table>
```

In the BAM files, the qStart, qEnd are contained in the qs and qe tags, (and reflected in the QNAME); the bounds of the aligned query in the ZMW read can be determined by adjusting qs and qe by the number of soft-clipped bases at the ends of the alignment (as found in the CIGAR).

See also the comments in the code for other tags.

Constructor

Parameters

- `filename (str)` -- filename of the input pacbio BAM file. The content of the BAM file is not the output of a mapper. Instead, it is the output of a Pacbio (Sequel) sequencing (e.g., subreads).
- `sample (int)` -- for sample, you can set the number of subreads to read (0 means read all subreads)

`boxplot_read_length_vs_passes(nmax=20, ax=None, whis=1.5, widths=0.6)`

**property df**

`filter_length(output_filename, threshold_min=0, threshold_max=inf)`

Select and Write reads within a given range

Parameters

- `output_filename (str)` -- name of output file
- `threshold_min (int)` -- minimum length of the reads to keep
- `threshold_max (int)` -- maximum length of the reads to keep

`get_mean_nb_passes(min_length=50, max_length=1500000)`

`get_number_of_ccs(min_length=50, max_length=15000)`

`hist_GC(bins=50, alpha=0.5, hold=False, fontsize=12, grid=True, xlabel='GC %', ylabel='#', label='', title=None)`

Plot histogram GC content

Parameters

- `bins (int)` -- binning for the histogram
- `alpha (float)` -- transparency of the histograms
- `hold (bool)` --
- `fontsize (int)` -- fontsize of the x and y labels and title.
- `grid (bool)` -- add grid or not
- `xlabel (str)` --
- `ylabel (str)` --
- `label (str)` -- label of the histogram (for the legend)
- `title (str)` --

```python
from sequana.pacbio import PacbioSubreads
from sequana import sequana_data
b = PacbioSubreads(sequana_data("test_pacbio_subreads.bam"))
b.hist_GC()
```

`hist_nb_passes(bins=None, alpha=0.8, hold=False, fontsize=12, grid=True, xlabel='Number of passes', logy=True, ec='k', ylabel='#', label='', title='Number of passes')`

Plot histogram of number of passes

2.12. References
Parameters

- **alpha** (*float*) -- transparency of the histograms
- **hold** (*bool*) --
- **fontsize** (*int*) --
- **grid** (*bool*) --
- **xlabel** (*str*) --
- **ylabel** (*str*) --
- **logy** (*bool*) -- use log scale on the y axis (default to True)
- **label** (*str*) -- label of the histogram (for the legend)
- **title** (*str*) --

```python
from sequana.pacbio import PacbioSubreads
from sequana import sequana_data
b = PacbioSubreads(sequana_data("test_pacbio_subreads.bam"))
b.hist_nb_passes()
```

**hist_read_length** (*bins=50, alpha=0.8, hold=False, fontsize=12, grid=True, xlabel='Read Length', ylabel='#', label='', title=None, logy=False, ec='k', hist_kwargs={})

Plot histogram Read length

Parameters

- **bins** (*int*) -- binning for the histogram
- **alpha** (*float*) -- transparency of the histograms
- **hold** (*bool*) --
- **fontsize** (*int*) --
- **grid** (*bool*) --
- **xlabel** (*str*) --
- **ylabel** (*str*) --
- **label** (*str*) -- label of the histogram (for the legend)
- **title** (*str*) --

```python
from sequana.pacbio import PacbioSubreads
from sequana import sequana_data
b = PacbioSubreads(sequana_data("test_pacbio_subreads.bam"))
b.hist_read_length()
```

**hist_snr** (*bins=50, alpha=0.5, hold=False, fontsize=12, grid=True, xlabel='SNR', ylabel='#', title='', clip_upper_SNR=30)

Plot histogram of the ACGT SNRs for all reads

Parameters

- **bins** (*int*) -- binning for the histogram. Note that the range starts at 0 and ends at clip_upper_SNR
- **alpha** (*float*) -- transparency of the histograms
• `hold (bool)` --
• `fontsize (int)` --
• `grid (bool)` --
• `xlabel (str)` --
• `ylabel (str)` --
• `title (str)` --

```python
from sequana.pacbio import PacbioSubreads
from sequana import sequana_data
b = PacbioSubreads(sequana_data("test_pacbio_subreads.bam"))
b.hist_snr()
```

`plot_GC_read_len(hold=False, fontsize=12, bins=[200, 60], grid=True, xlabel='GC %', ylabel='#',
cmap='BrBG')`

Plot GC content versus read length

**Parameters**

• `hold (bool)` --
• `fontsize (int)` -- for x and y labels and title
• `bins` -- a integer or tuple of 2 integers to specify the binning of the x and y 2D histogram.
• `grid (bool)` --
• `xlabel (str)` --
• `ylabel (str)` --

```python
from sequana.pacbio import PacbioSubreads
from sequana import sequana_data
b = PacbioSubreads(sequana_data("test_pacbio_subreads.bam"))
b.plot_GC_read_len(bins=[10, 10])
```

`random_selection(output_filename, nreads=None, expected_coverage=None, reference_length=None, read_lengths=None)`

Select random reads

**Parameters**

• `nreads` -- number of reads to select randomly. Must be less than number of available reads in the orignal file.
• `expected_coverage` --
• `reference_length` --

if expected_coverage and reference_length provided, nreads is replaced automatically.

**Note:** to speed up computation (if you need to call random_selection many times), you can provide the mean read length manually

```python
reset()
```

2.12. References
save_summary(filename)

**property stats**

return basic stats about the read length

**stride(output_filename, stride=10, shift=0, random=False)**

Write a subset of reads to BAM output

**Parameters**

- **output_filename** *(str)* -- name of output file
- **stride** *(int)* -- optionnal, number of reads to read to output one read
- **shift** *(int)* -- number of reads to ignore at the begining of input file
- **random** *(bool)* -- if True, at each step the read to output is randomly selected

**summary()**

**to_fasta(output_filename, threads=2)**

Export BAM reads into a Fasta file

**Parameters**

- **output_filename** -- name of the output file (use .fasta extension)
- **threads** *(int)* -- number of threads to use

**Note:** this executes a shell command based on samtools

**Warning:** this takes a few minutes for 500,000 reads

**to_fastq(output_filename, threads=2)**

Export BAM reads into FastQ file

### 2.12.16 Phred quality

Manipulate phred quality of reads

FastQ quality are stored as characters. The phred scales indicates the range of characters.

In general, characters goes from ! to ~ that is from 33 to 126 in an ascii table. This convention starts at 33 because characters before ! may cause trouble (e.g. white spaces). This scale is the Sanger scale. There are 2 other scales that could be used ranging from 59 to 126 (illumina 1) and from 64 to 126 (illumina 1.3+).

So, here are the offset to use:

<table>
<thead>
<tr>
<th>Name</th>
<th>offset</th>
<th>Numeric range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sanger</td>
<td>33</td>
<td>0 to 93</td>
</tr>
<tr>
<td>Solexa</td>
<td>64</td>
<td>-5 to 62</td>
</tr>
<tr>
<td>illumina1.3+</td>
<td>64</td>
<td>0 to 62</td>
</tr>
</tbody>
</table>

**reference** [http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2847217/](http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2847217/)
Even though dedicated tools would have better performances, we provide a set of convenient functions. An example is provided here below to plot the quality corresponding to a character string extracted from a FastQ read.

In this example, we use `Quality` class where the default offset is 33 (Sanger). We compare the quality for another offset.

```python
from sequana import phred

from sequana.phred import Quality
q = Quality('BCCFFFFFFHHHHIIJJJJJJIIJJJJJJJJFH')
q.plot()
q.offset = 64
q.plot()
from pylab import legend
legend(loc="best")
```

You can access to the quality as a list using the `quality` attribute and the mean quality from the `mean_quality` attribute.
property mean_quality
    return mean quality

plot(fontsize=16)
    plot quality versus base position

property quality
    phred string into quality list

proba_to_quality_sanger(pe)
    A value between 0 and 93

    Parameters pe -- the probability of error.
    Returns Q is the quality score.
    • a high probability of error (0.99) gives Q=0
    • q low proba of errors (0.05) gives Q = 13
    • q low proba of errors (0.01) gives Q = 20

quality_to_proba_sanger(quality)
    Quality to probability (Sanger)

2.12.17 RiboDesigner

Ribodesigner module

class RiboDesigner(fasta, gff, output_directory, seq_type='rRNA', max_n_probes=384, force=False, threads=4)
    Design probes for ribosomes depletion.

    From a complete genome assembly FASTA file and a GFF annotation file:
    • Extract genomic sequences corresponding to the selected seq_type.
    • For these selected sequences, design probes computing probe length and inter probe space according to the length of the ribosomal sequence.
    • Detect the highest cd-hit-est identity threshold where the number of probes is inferior or equal to max_n_probes.
    • Report the list of probes in BED and CSV files.

    In the CSV, the oligo names are in column 1 and the oligo sequences in column 2.

    Parameters
    • fasta -- The FASTA file with complete genome assembly to extract ribosome sequences from.
    • gff -- GFF annotation file of the genome assembly.
    • output_directory -- The path to the output directory.
    • seq_type -- string describing sequence annotation type (column 3 in GFF) to select rRNA from.
    • max_n_probes -- Max number of probes to design
    • force -- If the output_directory already exists, overwrite it.
    • threads -- Number of threads to use in cd-hit clustering.
cluster_probes()
Use cd-hit-est to cluster highly similar probes.

clustering_needed(force=False)
Checks if a clustering is needed.

Parameters
force -- force clustering even if unnecessary.

export_to_csv_bed()
Export final results to CSV and BED files

export_to_fasta()
From the self.probes_df, export to FASTA and CSV files.

get_all_probes()
Run all probe design and concatenate results in a single DataFrame.

get_rna_pos_from_gff()
Convert a GFF file into a pandas DataFrame filtered according to the self.seq_type.

run()

2.12.18 RNAdiff

class RNADesign(filename, sep=\n*,\n*, condition_col='condition', reference=None)
Simple RNA design handler

property comparisons

property conditions

keep_conditions(conditions)

class RNADiffAnalysis(counts_file, design_file, condition, keep_all_conditions=False, reference=None, comparisons=None, batch=None, fit_type='parametric', beta_prior=False, independent_filtering=True, cooks_cutoff=None, gff=None, fc_attribute=None, fc_feature=None, annot_cols=None, threads=4, outdir='rnadiff', sep_counts=';', sep_design=\n*,\n*, minimum_mean_reads_per_gene=0, minimum_mean_reads_per_condition_per_gene=0)
A tool to prepare and run a RNA-seq differential analysis with DESeq2

Parameters

• counts_file -- Path to tsv file out of FeatureCount with all samples together.
• design_file -- Path to tsv file with the definition of the groups for each sample.
• condition -- The name of the column from groups_tsv to use as condition. For more advanced design, a R function of the type 'condition*inter' (without the '~') could be specified (not tested yet). Each name in this function should refer to column names in groups_tsv.
• comparisons -- A list of tuples indicating comparisons to be made e.g A vs B would be [("A", "B")]
• batch -- None for no batch effect or name of a column in groups_tsv to add a batch effect.
• keep_all_conditions -- if user set comparisons, it means will only want to include some comparisons and therefore their conditions. Yet, sometimes, you may still want to keep all conditions in the differential analysis. If some set this flag to True.
- **fit_type** -- Default "parametric".
- **beta_prior** -- Default False.
- **independent_filtering** -- To let DESeq2 perform the independentFiltering or not.
- **cooks_cutoff** -- To let DESeq2 decide for the CooksCutoff or specifying a value.
- **gff** -- Path to the corresponding gff3 to add annotations.
- **fc_attribute** -- GFF attribute used in FeatureCounts.
- **fc_feature** -- GFF feature used in FeatureCounts.
- **annot_cols** -- GFF attributes to use for results annotations
- **threads** -- Number of threads to use
- **outdir** -- Path to output directory.
- **sep_counts** -- The separator used in the input count file.
- **sep_design** -- The separator used in the input design file.

This class reads a sequana.featurecounts.

```r
r = rnadiff.RNADiffAnalysis("counts.csv", "design.csv",
    condition="condition", comparisons=[(("A", "B"), ("A", "C")],
```

For developers: the rnadiff_template.R script behind the scene expects those attributes to be found in the RNADiffAnalysis class: counts_filename, design_filename, fit_type, fonction, comparison_str, independent_filtering, cooks_cutoff, code_dir, outdir, counts_dir, beta_prior, threads

- **check_and_save_input_tables** (sep_counts)
- **check_comparisons**()
- **check_condition**()
- **run**()

Create outdir and a DESeq2 script from template for analysis. Then execute this script.

Returns a RNADiffResults instance

```r
template = <Template 'rnadiff_light_template.R'>
```

class RNADiffResults(rnadiff_folder, gff=None, fc_attribute=None, fc_feature=None,
    pattern='*_degs_DESeq2.csv', alpha=0.05, log2_fc=0,
    palette=[(0.2345098039215686, 0.6274509803921567, 0.6674509803921569)],
    condition="condition", annot_cols=None, **kwargs)
```

The output of a RNADiff analysis

**Rnadiff_folder** a valid rnadiff folder created by RNADiffAnalysis
from sequana.rnadiff import RNADiffResults
from sequana import sequana_data
r = RNADiffResults(sequana_data("rnadiff/", "doc"))
r.plot_count_per_sample()

plot_dendogram(max_features=5000, transform_method='log', method='ward', metric='euclidean')

plot_density()

plot_dispersion()

plot_feature_most_present(fontsize=None, xticks_fontsize=None)

plot_isomap(n_components=2, colors=None)
  IN DEV, not functional

plot_mds(n_components=2, colors=None, clf=True)
  IN DEV, not functional
plot_pca(n_components=2, colors=None, plotly=False, max_features=500, genes_to_remove=[], fontsize=10, adjust=True)

from sequana.rnadiff import RNADiffResults
from sequana import sequana_data

r = RNADiffResults(sequana_data("rnadiff/", "doc"))

colors = {
    'surexp1': 'r',
    'surexp2': 'r',
    'surexp3': 'r',
    'surexp1': 'b',
    'surexp2': 'b',
    'surexp3': 'b'}
r.plot_pca(colors=colors)

plot_percentage_null_read_counts(fontsize=None, xticks_fontsize=None)

Bars represent the percentage of null counts in each samples. The dashed horizontal line represents the percentage of feature counts being equal to zero across all samples

from sequana.rnadiff import RNADiffResults
from sequana import sequana_data

r = RNADiffResults(sequana_data("rnadiff/", "doc"))
r.plot_percentage_null_read_counts()

plot_upset(force=False)

Plot the upset plot (alternative to venn diagram).

read_annot(gff)

Get a properly formatted dataframe from the gff.

Parameters gff -- a input GFF filename or an existing instance of GFF3

read_csv(filename)

report()

summary()

to_csv(filename)

class RNADiffTable(path, alpha=0.05, log2_fc=0, sep=';', condition='condition')

A representation of the results of a single rnadiff comparison

Expect to find output of RNADiffAnalysis file named after condt1_vs_cond2_degs_DESeq2.csv

from sequana.rnadiff import RNADiffTable
RNADiffTable("A_vs_B_degs_DESeq2.csv")

property alpha

filter()

filter a DESeq2 result with FDR and logFC thresholds
property log2_fc
plot_padj_hist(bins=60, fontsize=16)
plot_pvalue_hist(bins=60, fontsize=16, rotation=0)
plot_volcano(padj=0.05, add_broken_axes=False, markersize=4, limit_broken_line=[20, 40], plotly=False, annotations=None, hover_name=None)

```
from sequana.rnadiff import RNADiffResults
from sequana import sequana_data
r = RNADiffResults(sequana_data("rnadiff/", doc))
r.comparisons["A_vs_B"].plot_volcano()
```

def set_gene_lists()


def summary()

class RNADiffCompare(r1, r2=None, r3=None, design=None)
An object representation of results coming from a RNADiff analysis.

```
from sequana.compare import RNADiffCompare
c = RNADiffCompare("data.csv", "data2.csv")
```

plot_common_major_counts(mode, labels=None, switch_up_down_cond2=False, add_venn=True, xmax=None, title='"', fontsize=12, sortby='log2FoldChange')

Parameters

- **mode** -- down, up or all

```
from sequana import sequana_data
from sequana.compare import RNADiffCompare
c = RNADiffCompare(
    sequana_data("rnadiff_salmon.csv", "doc/rnadiff_compare"),
    sequana_data("rnadiff_bowtie.csv", "doc/rnadiff_compare")
)
c.plot_common_major_counts("down")
```

plot_corrplot_counts_normed(samples=None, log2=True, lower='pie', upper='text')
plot_corrplot_counts_raw(samples=None, log2=True, lower='pie', upper='text')
plot_foldchange()
plot_jaccard_distance(mode, padjs=[0.0001, 0.001, 0.01, 0.05, 0.1], Nfc=50, smooth=False, window=5)
plot_venn_all(labels=None, ax=None, title='all expressed genes', mode='all')
plot_venn_down(labels=None, ax=None, title='Down expressed genes', mode='all')
plot_venn_up(labels=None, ax=None, title='Up expressed genes', mode='all')

Venn diagram of cond1 from RNADiff result1 vs cond2 in RNADiff result 2
from sequana import sequana_data
from sequana.compare import RNADiffCompare

c = RNADiffCompare(
    sequana_data("rnadiff_salmon.csv", "doc/rnadiff_compare"),
    sequana_data("rnadiff_bowtie.csv", "doc/rnadiff_compare")
)
c.plot_venn_up()

plot_volcano(labels=None)
Volcano plot of log2 fold change versus log10 of adjusted p-value

plot_volcano_differences(mode='all')

2.12.19 Running median

Data analysis tool

RunningMedian(data, width[, container]) Running median (fast)

class RunningMedian(data, width, container=<class 'list'>)
    Running median (fast)
    
    This is an efficient implementation of running median, faster than SciPy implementation v0.17 and a skip list method.
    
    The main idea comes from a recipe posted in this website: http://code.activestate.com/recipes/576930/#c3 that uses a simple list as proposed in https://gist.github.com/f0k/2f8402e4dfb6974bfcf1 and was adapted to our needs included object oriented implementation.

    Note: a circular running median is implemented in sequana.bedtools.GenomeCov

    from sequana.running_median import RunningMedian
    rm = RunningMedian(data, 101)
    results = rm.run()

    Warning: the first W/2 and last W/2 positions should be ignored since they do not use W values. In this implementation, the last W/2 values are currently set to zero.

    This shows how the results agree with scipy

2.12. References
from pylab import *
import scipy.signal
from sequana.running_median import RunningMedian

clf()
x = randn(100)
plot(x, 'k')
plot(RunningMedian(x, 9).run(), 'r', lw=4)
plot(scipy.signal.medfilt(x, 9), 'go')
grid()
Note that for visualisation, we set the ylimits to 50 but the data at position 500 goes up to 120 and there is an
large outlier (1000) at position 700.

We see that the median is less sensible to the outliers, as expected. The median is highly interesting for large
outliers on short duration (e.g. here the peak at position 500) but is also less biases by larger regions.

Note: The beginning and end of the running median are irrelevant. There are actually equal to the data in our
implementation.
Note: using blist instead of list is not always faster. It depends on the width of the window being used. list and blist are equivalent for W below 20,000 (list is slightly faster). However, for large W, blist has an $O(\log(n))$ complexity while list has a $O(n)$ complexity.

**constructor**

Parameters

- **data** -- your data vector
- **width** -- running window length
- **container** -- a container (defaults to list). Could be a B-tree blist from the blist package but is 30% slower than a pure list for $W < 20,000$

scipy in $O(n)$ list in $\sqrt{n}$ blist in $O(\log(n))$

```python
run()
```

```python
running_median(data, width, container=<class 'list'>)
```

### 2.12.20 Snpeff module

Tools to launch snpEff.

**class SnpEff(***annotation*, **log=None**, **snpeff_datadir='data'**, **fastafile=None**)**

SnpEff is a tool dedicated to annotate detected variants in a VCF file. This wrapper eases the annotation with a genbank file. It create automatically the custom database. Then, run snpEff with a subprocess. Caution, the locus name (or chromosome name) in genbank file and the sequence name in VCF file must be the same. Otherwise, snpEff is not able to bind informations.

Example:

```python
snpeff = SnpEff('file.gbk')
snpeff.launch_snpeff('variants.vcf', 'variant.ann.vcf')
```

If your input is in GFF format, you must also provide the fasta reference file.

Will save relevant snpeff data into ./data directory (or snpeff_datadir).

**Constructor**

Parameters

- **annotation** -- annotation reference.
- **file_format** -- format of your file. ('only genbank actually')
- **log** -- log file
- **snpeff_datadir** -- default to data.
- **fastafile** -- if a GFF is used, you must provide the FASTA input file as well
add_locus_in_fasta(fasta, output_file)
Add locus of annotation file in description line of fasta file. If fasta file and genbank file do not have the same names.

Parameters
- fasta (str) -- input fasta file where you want to add locus.
- output_file (str) -- output file.

FIXME: fasta is already known if provided in the init

launch_snpeff(vcf_filename, output, html_output=None, options="")
Launch snpEff with the custom genbank file.

Parameters
- vcf_filename (str) -- input VCF filename.
- output (str) -- output VCF filename.
- html_output (str) -- filename of the HTML creates by snpEff.
- options (str) -- any options recognised by snpEff.

download_fasta_and_genbank(identifier, tag, genbank=True, fasta=True, outdir='.')

Parameters
- identifier -- valid identifier to retrieve from NCBI (genbank) and ENA (fasta)
- tag -- name of the filename for the genbank and fasta files.

2.12.21 General tools

calc utils

findpos(seq, chr)
Find position(s) of a substring into a longer string.
Note that this function is a generator:

```python
>>> list(findpos("AACCGGAAGGTT", "GG"))
[4, 8]
```

normpdf(x, mu, sigma)
Return the normal pdf evaluated at x; args provides mu, sigma

Note: same as scipy.stats.norm but implemented to avoid scipy dependency

texwrap(text, width=80, indent=0)
Wrap a string with 80 characters

Parameters
- text -- input text
- width -- (defaults to 80 characters)
- indent -- possible indentation (0 by default)
wget(link, output)
Retrieve a file from internet.

Parameters
- link(str) -- a valid URL
- output(str) -- the output filename

Warning: no sanity check of any kind for now

Todo: move to easydev

General tools

class GZLineCounter(filename)
Fast GZipped line counter
Uses zcat if possible, otherwise gzip library (twice as slow).

```python
>>> from sequana import sequana_data
>>> from sequana.tools import GZLineCounter
>>> gz = GZLineCounter(sequana_data("test.fastq.gz"))
>>> len(gz)
1000
```

class StatsBAM2Mapped(bamfile=None, wkdir=None, verbose=True)

```
    to_html()
    to_json(filename)
```

bam_to_mapped_unmapped_fastq(filename, output_directory=None, verbose=True)
Create mapped and unmapped fastq files from a BAM file

Context given a reference, one or two FastQ files are mapped onto the reference to generate a BAM file. This BAM file is a compressed version of a SAM file, which interpretation should be eased within this function.

Parameters
- filename -- input BAM file
- output_directory -- where to save the mapped and unmapped files

Returns dictionary with number of reads for each file (mapped/unmapped for R1/R2) as well as the mode (paired or not), the number of unpaired reads, and the number of duplicated reads. The unpaired reads should be zero (sanity check)

Given a BAM file, create FASTQ with R1/R2 reads mapped and unmapped. In the paired-end case, 4 files are created.

Note that this function is efficient in that it does not create intermediate files limiting IO in the process. As compared to standard tools such as bedtools bamtofastq, it is 1.5 to 2X slower but it does create the mapped AND unmapped reads.
Details  Secondary alignment (flag 256) are dropped so as to remove any ambiguous alignments. The output dictionary stores "secondary" key to keep track of the total number of secondary reads that are dropped. If the flag is 256 and the read is unpaired, the key unpaired is also incremented.

If the flag is not equal to 256, we first reverse complement reads that are tagged as reverse in the BAM file. Then, reads that are not paired or not "proper pair" (neither flag 4 nor flag 8) are ignored.

If R1 is mapped or R2 is mapped then the reads are considered mapped. If both R1 and R2 are unmapped, then reads are unmapped.

Note:  about chimeric alignment: one is the representative and the other is the supplementary. This flag is not used in this function. Note also that chimeric alignment have same QNAME and flag 4 and 8

Note:  the contamination reported is based on R1 only.

Todo:  comments are missing since there are not stored in the BAM file.

Note:  the mapped reads may not be synchronized because we include also the chimeric alignment (cf samtools documentation). However, total reads = unmappeds reads + R1 mapped + R2 mapped - supplementary reads (those with flag 2048).

2.12.22 Format IO

class BED(filename)  
a structure to read and manipulate BED files (12-column file)

columns are defined as chromosome name, start and end, gene_name, score, strand, CDS start and end, block count, block sizes, block starts:

get_CDS_exons()  
Extract CDS from input BED file.

get_exons()  
Extract exon regions from input BED file.

Uses the first (chromosome name), second (chromosome start), 11th and 12th columns (exon start and size) of a 12-columns BED file.

```python
from sequana import sequana_data
from sequana import BED
b = BED(sequana_data("hg38_chr18.bed"))
b.get_exons()
```

get_transcript_ranges()  
Extract transcript from input BED file.
class GFF3(filename, skip_types=['biological_region'])

Read a GFF file, version 3

See also:
https://github.com/The-Sequence-Ontology/Specifications/blob/master/gff3.md

```python
g = GFF3(filename)
# first call is slow
g.df
# print info about the different feature types
g.features
# prints info about duplicated attributes:
g.get_duplicated_attributes_per_genetic_type(self)
```

On eukaryotes, the reading and processing of the GFF may take a while. On prokaryotes, it should be pretty fast (a few seconds). To speed up the eukaryotes case, we skip the processing biological_regions (50% of the data in mouse).

**property attributes**

Return list of possible attributes

If feature is provided, must be valid and used as a filter to keep only entries for that feature.

~10 seconds on mouse genome GFF file.

**create_files_for_rnadiff**

```python
create_files_for_rnadiff(outname, genetic_type='gene', ID='Name', fields=['Name'],
merge_identical_id=True)
```

Creates two files required for the RNADiff analysis following sequana_rnaseq pipeline

**Parameters**

- **outname (str)** -- the output filename prefix
- **genetic_type** -- genetic type to be selected from the GFF file e.g. gene (default), CDS, etc
- **ID** -- the identifier (key) to be selected from the list of attributes found in the GFF for the given type. By default, 'Name'. Used as first column in the two output file.
- **fields** -- the fields to be save in the outname_info.tsv file
- **merge_identical_id** -- it may happen that the same gene Name has two entries (e.g in e-coli with 2 unique IDs have the same name with an annotation such as partI and part II). If so, feature counts is clever enough to deal with it. Here, we need to merge the entries and sum the length together. Ideally, one should not use the Name but ID or gene_id or locus_tag.

**Returns** nothing

This functions reads the GFF file and creates two files:

1. outname_gene_lengths.tsv contains column 1 with identifiers and column 2 with length of the selected type (e.g. gene)
2. outname_info.tsv first column is the same identifier as in the first file and following columns contain the fields of interest (Name by default but could be any attributes to be found in the GFF such as description

**property df**
**property_features**

Extract unique GFF feature types

This is equivalent to awk '{print $3} | sort | uniq' to extract unique GFF types. No sanity check, this is suppose to be fast.

Less than a few seconds for mammals.

**get_attributes(feature=None, sep=';')**

Return list of possible attributes

If feature is provided, must be valid and used as a filter to keep only entries for that feature.

~10 seconds on mouse genome GFF file.

**get_duplicated_attributes_per_genetic_type()**

**read()**

Read annotations one by one creating a generator

**read_and_save_selected_features(outfile, features=['gene'])**

**save_annotation_to_csv(filename='annotations.csv')**

**save_gff_filtered("test.gff", features=['misc_RNA', 'rRNA'], replace_seqid='locus_tag')**

**to_bed(output_filename, attribute_name)**

Experimental export to BED format to be used with rseqc scripts

**Parameters**

- **attribute_name** *(str)* -- the attribute name to be found in the GFF attributes

**to_fasta(ref_fasta, fasta_out)**

From a genomic FASTA file ref_fasta, extract regions stored in the gff. Export the corresponding regions to a FASTA file fasta_out.

**Parameters**

- **ref_fasta** -- path to genomic FASTA file to extract rRNA regions from.
- **fasta_out** -- path to FASTA file where rRNA regions will be exported to.

**to_gtf(output_filename='test.gtf', mapper={'ID': '/_id'})**

**transcript_to_gene_mapping(feature='all', attribute='transcript_id')**

**Parameters**

- **feature** -- not used yet
- **attribute** -- the attribute to be used. should be transcript_id for salmon compatibility but could use something different.
2.12.23 VCF module

Analysis of VCF file generated by freebayes.

```python
class Filtered_freebayes(variants, fb_vcf):
    Variants filtered with VCF_freebayes.

    constructor
    Parameters
      • variants (list) -- list of variants record.
      • fb_vcf (VCF_freebayes) -- class parent.

    property columns
        Get columns index.

    property df
        Get the data frame.

to_csv(output_filename, info_field=False)
    Write DataFrame in CSV format.
        Params str output_filename output CSV filename.

to_vcf(output_filename)
    Write VCF file in VCF format.
        Params str output_filename output VCF filename.

    property variants
        Get the variant list.

    property vcf
        Get the VCF_freebayes object.
```

```python
class VCF_freebayes(filename, **kwargs):
    VCF class (Variant Calling Format)
    This class is a wrapping of vcf.Reader class from the pyVCF package. It is dedicated for VCF file generated by
    freebayes. A data frame with all variants is produced which can be written as a csv file. It can filter variants with
    a dictionary of filter parameter. Filter variants are written in a new VCF file.
```

```python
from sequana import sequana_data
from sequana.freebayes_vcf_filter import VCF_freebayes
vcf_filename = sequana_data("JB409847.vcf")

# Read the data
v = VCF_freebayes(vcf_filename)

# Filter the data
filter_dict = {"freebayes_score": 200,
               "frequency": 0.8,
               "min_depth": 10,
               "forward_depth": 3,
               "reverse_depth": 3,
```
Information about strand bias (aka strand balance, or strand ratio). This is a type of sequencing bias in which one DNA strand is favored over the other, which can result in incorrect evaluation of the amount of evidence observed for one allele vs. the other.

**constructor**

Parameters

- **filename** (str) -- a vcf file.
- **kwargs** -- any arguments accepted by vcf.Reader

**filter_vcf** *(filter_dict=None)*

Filter variants in the VCF file.

Parameters

- **filter_dict** (dict) -- dictionary of filters. It updates the attribute `VCF_freebayes.filter_params`

Return Filtered_freebayes object.

**property filters_params**

Get or set the filters parameters to select variants of interest. Setter take a dictionnary as parameter to update the attribute `VCF_freebayes.filters_params`. Delete will reset different variable to 0.

```python
v = VCF_freebayes("input.vcf")
v.filters_params = {
    "freebayes_score": 200,
    "frequency": 0.8,
    "min_depth": 10,
    "forward_depth": 3,
    "reverse_depth": 3,
    "strand_ratio": 0.2
}
```

**get_variants**

**property is_joint**

Get `VCF_freebayes.is_joint` if the vcf file is a joint_freebayes.

**rewind**

Rewind the reader

**property samples**

**class Variant(record)**

Variant reader and dictionary that stores important variant information
constructor

Parameters

- `record` ([RecordVariant]) -- variant record
- `resume` ([dict]) -- most important informations of variant

property record
property resume
property samples

2.12.24 Module Reports

Generic module is the parent module of all other module

class SequanaBaseModule(templat_fn='standard.html', required_dir=None)

Generic Module to write HTML reports.

# to add a TOC, add this code:

```html
<div id="tocDiv">
  <ul id="tocList"></ul>
</div>
```

add_code_section(content, language)

Add code in your html.

add_float_right(content)

Align a content to right.

add_fotorama(files, width=600, height=800, loop=True, thumbnails=True, file_thumbnails=None, captions=None)

Copy a file to a target directory in report dir. Return the relative path of your file.

Parameters

- `filename` ([str]) -- file to copy.
- `target_dir` ([str]) -- directory where to copy.

Return relative path of the new file location.

create_combobox(path_list, html_id, newtab=True)

Create a dropdown menu with QueryJS.

Parameters `path_list` ([list]) -- list of links.

return html div and js script as string.

create_embedded_png(plot_function, input_arg, style=None, **kwargs)

Take as a plot function as input and create a html embedded png image. You must set the arguments name for the output to connect buffer.
create_hide_section(html_id, name, content, hide=False)
Create an hideable section.

Parameters
- html_id (str) -- add short id to connect all elements.
- name (str) -- name of the hyperlink to hide or display the content.
- content (str) -- hideable HTML content.
- hide (bool) -- set if the first state is hiding or not.

Return tuple that contains HTML hyperlink and hideable section.

create_html(output_filename)
Create HTML file with Jinja2.

Parameters output_filename (str) -- HTML output filename

create_link(name, target, newtab=True, download=False)
Create an HTML hyperlink with name and target.

Parameters
- target (str) -- the target url.
- newtab (bool) -- open html page in a new tab.
- download (bool) -- download the target.

Return as string the HTML hyperlink to the target.

include_svg_image(filename, alt=’undefined’)
Include SVG image in the html.

png_to_embedded_png(png, style=None, alt=’’)
Include a PNG file as embedded file.

Report dedicated to BAM file

BAMQCModule(bam_input[, output_filename]) Report dedicated to BAM file

class BAMQCModule(bam_input, output_filename=None)
Report dedicated to BAM file

    from sequana import sequana_data
    from sequana.modules_report.bamqc import BAMQCModule
    filename = sequana_data("test.bam")

    r = BAMQCModule(filename)
    r.create_html("test.html")

    # report/bam.html is now available

Todo: right now, the computation is performed in the class. Ideally, we would like the computation to happen elsewhere, where a json is stored. The json would be the input to this class.
add_flag_section()
add_images_section()
create_report_content()

Module to write coverage report

class ChromosomeCoverageModule(chromosome, datatable, region_window=200000, options=None, command="")

Write HTML report of coverage analysis for each chromosome. It is created by CoverageModule.

Parameters

- chromosome --
- datatable --
- directory --
- region_window (int) -- length of the sub coverage plot
- options -- should contain "W", "k", "circular"

add_command()

basic_stats()

Basics statistics section.

coverage_barplot()

Coverage barplots section.

coverage_plot()

Coverage section.

create_report_content(directory, options=None)

Generate the sections list to fill the HTML report.

gc_vs_coverage()

3 dimensional plot of GC content versus coverage.

normalized_coverage()

Barplot of normalized coverage section.

regions_of_interest(rois, links)

Region of interest section.

subcoverage(rois, directory)

Create subcoverage reports to have access to a zoomable line plot.

Params rois

Parameters directory -- directory name for the chromosome

This method create sub reports for each region of 200,000 bases (can be changed). Usually, it starts at position 0 so reports will be stored in e.g. for a genome of 2,300,000 bases:

```
chromosome_name/chromosome_name_0_200000.html
chromosome_name/chromosome_name_200000_400000.html
...
```
Note that if the BED file positions does not start at zero, then names will take care of that.

```
zscore_distribution()
```

Barplot of zscore distribution section.

```
class CoverageModule(data, region_window=200000)
```

Write HTML report of coverage analysis. This class takes either a genomecov instances or a csv file where analysis are stored.

```
constructor

Parameters

- **data** -- it can be a csv filename created by sequana_coverage or a bedtools.GenomeCov object.
- **region_window** --

create_chromosome_reports()

Create HTML report for each chromosome present in data.

create_chromosome_table(html_list)

Create table with links to chromosome reports

create_report_content(html_list)

init_roi_datatable()

Initiate DataTableFunction to create table to link each row with sub HTML report. All table will have the same appearance. We can therefore initialise the roi once for all.

```
Parameters rois -- can be a ROIs from ChromosomeCov instance or a simple dataframe
```

### 2.12.25 Wrapper to other tools

```
class ITOL(tree, APIkey=None, projectName=None)
```

Tree with branch lengths:

```
(A:0.1,(B:0.1,C:0.1));
```

Tree with bootstrap and branch lengths:

```
(A:0.1,(B:0.1,C:0.1)90:0.1)98:0.3);
```

```
from pylab import imshow, imread
from easydev import TempFile
from sequana import ITOL, sequana_data
```

```
# You must have an APIkey and project name defined on itol web site.
itol = ITOL(sequana_data("test_itol_basic.tree.txt"), APIkey, projectName)
itol.upload()
```

(continues on next page)
# You can change the parameters in itol.params
itol.params["display_mode"] = 1  # use linear layout instead of circular

# finally export your image locally:
with TempFile(suffix=".png") as fout:
    itol.export(fout.name)
imshow(imread(fout.name))

For details, please see https://itol.embl.de/help.cgi#annot Here are some parameters:

- display_mode: 1, 2 or 3 (1 = rectangular, 2 = circular, 3 = unrooted)

**constructor**

**add_file**(filename)

**export**(filename='test.png', extra_params={}, tree_id=None, circular=True)

Export or retrieve an existing tree to get back the resulting image

**Parameters**

- **filename**(str) -- the filename where to store the image
- **extra_params** -- parameters use to tune the tre are stored in params but you may provide extra parameters here ot alter some. If you this paramterm, the main attribute params is unchanged.
- **tree_id** -- if you have a known tree identifier, you can retrieve it using the tree_id parameter. Otherwise, if you have just uploaded a tree with upload(), the identifier is automatically populated and that is the tree you will export.

**upload()**

**class CNVnator**(filename)

Reader of the CNVnator output file.

**hist_event_size**(bins=20)

**plot**(chr_name, x1=None, x2=None, Y=20)

**class CanuScanner**(path=':'))

**getfile**(filename)

**hist_read_length**(bins=100, fontsize=16)

**hist_read_length2**(fontsize=16)

**hist_trimming_read_length**(bins=100, fontsize=16)

**plot_correction_check1**(alpha=0.5)

**plot_kmer**(bins=100)

**scan_correction()**

**scan_trimming()**
set_overlap_filtering()

set_read_correction()

class StatsFile(filename='Stats.json')
    Reads a bcl2fastq Stats.json file and produces useful plots for QCs

barplot(filename='lane{}_status.png', lanes=None)

barplot_per_sample(alpha=0.5, width=0.8, filename=None)

barplot_summary(filename=None, color=['green', 'red'], alpha=0.8)

get_data_reads()

plot_unknown_barcodes(N=20)

to_summary_reads(filename)

class TRF(filename, verbose=False, frmt=None)
    Tandem Repeat Finder utilities

    The input data is the output of trf tool when using the -d option. This is not a CSV file. It contains comments in the middle of the file to indicate the name of the contig.

    The output filename has the following filename convention:

    ```
    test.fa.2.5.7.80.10.50.2000.dat
    ```

    where the numbers indicate the 7 input parameters:
    - Match = matching weight
    - Mismatch = mismatching penalty
    - Delta = indel penalty
    - PM = match probability (whole number)
    - PI = indel probability (whole number)
    - Minscore = minimum alignment score to report
    - MaxPeriod = maximum period size to report

    You may use `-h` to suppress html output.

    Then, you can use this class to easily identify the pattern you want:

    ```python
    t = TRF("input.dat")
    query = "length>100 and period_size==3 and entropy>0 and C>20 and A>20 and G>20"
    t.df.query(query)
    ```

    hist_cnvs(bins=50, CNVmin=10, motif=['CAG', 'AGC', 'GCA'], color='r', log=True)
        histogram of the motif found in the list provided by users. As an example, this is triplet CAG. Note that we also add the shifted version AGC and GCA.

    hist_entropy(bins=50)
        Histogram of the entropy of all found repeats

2.12. References
**hist_length_repetition**(*bins=50, CNVmin=3, motif=['CAG', 'AGC', 'GCA'], color='r', log=True*)

Histogram of the motif found in the list provided by users. As an example, this is triplet CAG. Note that we also add the shifted version AGC and GCA.

**hist_period_size**(*bins=50*)

Length of the repetitions

**hist_repet_by_sequence**()

**scandata**(*verbose=True, max_seq_length=20*)

Scan output of trf and returns a dataframe

The format of the output file looks like:

```
Tandem Repeats Finder Program
some info
Sequence: chr1
Parameters: 2 5 7 80 10 50 2000

10001 10468 6 77.2 6 95 3 801 33 51 0 15 1.43 TAACCC TAACCCTA...
1 10 6 77.2 6 95 3 801 33 51 0 15 1.43 TAACCC TAACCCTA...

Sequence: chr2
Parameters: 2 5 7 80 10 50 2000

10001 10468 6 77.2 6 95 3 801 33 51 0 15 1.43 TAACCC TAACCCTA...
```

The dataframe stores a row for each sequence and each pattern found. For instance, from the example above you will obtain 3 rows, two for the first sequence, and one for the second sequence.

---

### 2.12.26 Misc

Retrieve data from sequana library

**sequana_data**(filename=None, where=None)

Return full path of a sequana resource data file.

**Parameters**

- *filename* (str) -- a valid filename to be found
- *where* (str) -- one of the registered data directory (see below)

**Returns** the path of file. See also here below in the case where filename is set to "*".

```python
from sequana import sequana_data
filename = sequana_data('test.fastq')
```

Type the function name with "*" parameter to get a list of available files. With the where argument set, the function returns a list of files. Without the where argument, a dictionary is returned where keys correspond to the registered directories:
filenames = sequana_data("*", where="images")

Registered directories are:
- data
- testing
- images

Note: this does not handle wildcards. The * means retrieve all files.

Some useful data sets to be used in the analysis

The command `sequana.sequana_data()` may be used to retrieved data from this package. For example, a small but standard reference (phiX) is used in some NGS experiments. The file is small enough that it is provided within sequana and its filename (full path) can be retrieved as follows:

```python
from sequana import sequana_data
cfullpath = sequana_data("phiX174.fa", "data")
```

Other files stored in this directory will be documented here.

```python
adapters = {
    'adapters_netflex_pcr_free_1_fwd': 'adapters_netflex_pcr_free_1_fwd.fa',
    'adapters_netflex_pcr_free_1_rev': 'adapters_netflex_pcr_free_1_rev.fa'}
```

List of adapters used in various sequencing platforms

Sniffer

```python
sniffer(filename)
```

Utilities to create a Jquery DataTable for your HTML file.

| `DataTableFunction(df, html_id[, index])` | Class that contains Jquery DataTables function and options. |
| `DataTable(df, html_id[, datatable, index])` | Class that contains html table which used a javascript function. |

```python
class DataTable(df, html_id, datatable=None, index=False)
```

Class that contains html table which used a javascript function.

You must add in your HTML file the JS function (`DataTable.create_javascript_function()`) and the HTML code (`DataTable.create_datatable()`).

Example:

```python
df = pandas.read_csv('data.csv')
datatable = DataTable(df, 'data')
datatable.datatable.datatable_options = {
    'pageLength': 15,
    'dom': 'Bfrtip',
    'buttons': ['copy', 'csv']}
js = datatable.create_javascript_function()
html = datatable.create_datatable()
```

# Second CSV file with same format

```python
df2 = pandas.read_csv('data2.csv')
```

(continues on next page)
The reason to include the JS manually is that you may include many HTML table but need to include the JS only once.

**Constructor**

Parameters

- **df** -- data frame.
- **html_id** *(str)* -- the unique ID used in the HTML file.
- **datatable** *(DataTableFunction)* -- javascript function to create the Jquery Datatables. If None, a *DataTableFunction* is generated from the df.
- **index** *(bool)* -- indicates whether the index dataframe should

**create_datatable(style='width:100%';,****kwargs)**

Return string well formated to include in a HTML page.

Parameters

- **style** *(str)* -- CSS option of your table.
- **kwargs** *(dict)* -- parameters of *pandas.DataFrame.to_csv()*.

**create_javascript_function()**

Generate the javascript function to create the DataTable in a HTML page.

**property df**

**property html_id**

class DataTableFunction(df, html_id, index=False)

Class that contains Jquery DataTables function and options.

Example:

```python
import pandas as pd
from sequana.utils.datatables_js import DataTableFunction

df = pandas.read_csv('data.csv')
datatable_js = DataTableFunction(df, 'data')
datatable_js.datatable_options = {'pageLength': 15, 'dom': 'Bfrtip', 'buttons': ['copy', 'csv']}
js = datatable_js.create_javascript_function()
html_datatables = [DataTable(df, "data_{0}".format(i), datatable_js) for i, df in enumerate(df_list)]
```

Here, the datatable_options dictionary is used to fine tune the appearance of the table.

**Note:** DataTables add a number of elements around the table to control the table or show additional information about it. There are controlled by the order in the document (DOM) defined as a string made of letters, each of
them having a precise meaning. The order of the letter is important. For instance if B is first, the buttons are put before the table. If B is at the end, it is shown below the table. Here are some of the valid letters and their meaning:

- B: add the Buttons (copy/csv)
- i: add showing 1 to N of M entries
- f: add a search bar (f filtering)
- r: processing display element
- t: the table itself
- p: pagination control

Each option can be specified multiple times (with the exception of the table itself).

Note: other useful options are:

- pageLength: 15
- scrollX: "true"
- paging: 15
- buttons: ['copy', 'csv']

Note that buttons can also be excel, pdf, print, ...

All options of datatable: https://datatables.net/reference/option/

constructor

Parameters

- df -- data frame.
- html_id (str) -- the ID used in the HTML file.

create_javascript_function()

Return javascript to create the DataTable.

property datatable_columns

Get datatable_columns dictionary. It is automatically set from the dataframe you want to plot.

property datatable_options

Get, set or delete the DataTable options. Setter takes a dict as parameter with the desired options and updates the current dictionary.

Example:

```python
datatable = DataTableFunction("tab")
datatable.datatable_options = {'dom': 'Bfrtip',
                               'buttons': ['copy', 'csv']}
```

source: https://datatables.net/reference/option/

2.12. References
property html_id
Get the html_id, which cannot be set by the user after the instanciation of the class.

set_links_to_column(link_col, target_col, new_page=True)
Hide a column with urls and connect it with a column.

Parameters
• link_col (str) -- column with your URLs.
• target_col (str) -- column to connect.

set_tooltips_to_column(tooltips_col, target_col)
Hide a column with tooltips and connect it with a column.

Parameters
• tooltips_col (str) -- column with your tooltips.
• target_col (str) -- column to connect.

2.13 References (Viz)

Contents
• References (Viz)
  – Visualisation
    * Anova
    * corrplot
    * Heatmap, dendogram
    * Hinton plot
    * 2D Histogram
    * Image
    * PCA
    * ScatterPlot
    * Venn diagram
    * Volcano plots
2.13.1 Visualisation

Anova

class ANOVA(df)
    DRAFT
    Testing if 3(+) population means are all equal.
    Looks like the group are different, visually, and naively.

    from pylab import *
    from sequana.viz import ANOVA
    import pandas as pd

    A = normal(0.5, size=10000)
    B = normal(0.25, size=10000)
    C = normal(0, 0.5, size=10000)
    df = pd.DataFrame({"A":A, "B":B, "C":C})
    a = ANOVA(df)
    print(a.anova())
    a.imshow_anova_pairs()
Perform one-way ANOVA.

The one-way ANOVA tests the null hypothesis that two or more groups have the same population mean. The test is applied to samples from two or more groups. Since we are using a dataframe, vector length are identical.

return: the F value (test itself), and its p-value

```python
imshow_anova_pairs(log=True, **kargs)
```

corrplot

Corrplot utilities

references http://cran.r-project.org/web/packages/corrplot/vignettes/corrplot-intro.html

class Corrplot(data, na=0, compute_correlation=False)

An implementation of correlation plotting tools (corrplot)

Here is a simple example with a correlation matrix as an input (stored in a pandas dataframe):

```python
# create a correlation-like data set stored in a Pandas' dataframe.
import string
# letters = string.uppercase[0:10] # python2
letters = string.ascii_uppercase[0:10]
import pandas as pd
df = pd.DataFrame(dict(( (k, np.random.random(10)+ord(k)-65) for k in letters)))

# and use corrplot
from sequana.viz import corrplot
c = corrplot.Corrplot(df)
c.plot()
```

See also:

All functionalities are covered in this notebook

Constructor

Plots the content of square matrix that contains correlation values.

Parameters

- **data** -- input can be a dataframe (Pandas), or list of lists (python) or a numpy matrix. Note, however, that values must be between -1 and 1. If not, or if the matrix (or list of lists) is not squared, then correlation is computed. The data or computed correlation is stored in `df` attribute.
- **compute_correlation** (`bool`) -- if the matrix is non-squared or values are not bounded in -1,+1, correlation is computed. If you do not want that behaviour, set this parameter to False. (True by default).
- **na** -- replace NA values with this value (default 0)

The `params` contains some tunable parameters for the colorbar in the `plot()` method.
# can be a list of lists, the correlation matrix is then a 2x2 matrix
c = corrplot.Corrplot([[1,1], [2,4], [3,3], [4,4]])

df

The input data is stored in a dataframe and must therefore be compatible (list of lists, dictionary, matrices...)

order (method='complete', metric='euclidean', inplace=False)

Rearrange the order of rows and columns after clustering

Parameters

- **method** -- any scipy method (e.g., single, average, centroid, median, ward). See scipy.cluster.hierarchy.linkage
- **metric** -- any scipy distance (euclidean, hamming, jaccard) See scipy.spatial.distance or scipy.cluster.hierarchy
- **inplace** (bool) -- if set to True, the dataframe is replaced

You probably do not need to use that method. Use plot() and the two parameters order_metric and order_method instead.

params

tunable parameters for the plot() method.

plot(fig=None, grid=True, rotation=30, lower=None, upper=None, shrink=0.9, facecolor='white', colorbar=True, label_color='black', fontsize='small', edgecolor='black', method='ellipse', order_method='complete', order_metric='euclidean', cmap=None, ax=None, binarise_color=False)

plot the correlation matrix from the content of df (dataframe)

By default, the correlation is shown on the upper and lower triangle and is symmetric wrt to the diagonal. The symbols are ellipses. The symbols can be changed to e.g. rectangle. The symbols are shown on upper
and lower sides but you could choose a symbol for the upper side and another for the lower side using the `lower` and `upper` parameters.

**Parameters**

- **fig** -- Create a new figure by default. If an instance of an existing figure is provided, the corrplot is overlayed on the figure provided. Can also be the number of the figure.
- **grid** -- add grid (Defaults to grey color). You can set it to False or a color.
- **rotation** -- rotate labels on y-axis
- **lower** -- if set to a valid method, plots the data on the lower left triangle
- **upper** -- if set to a valid method, plots the data on the upper left triangle
- **shrink (float)** -- maximum space used (in percent) by a symbol. If negative values are provided, the absolute value is taken. If greater than 1, the symbols will overlap.
- **facecolor** -- color of the background (defaults to white).
- **colorbar** -- add the colorbar (defaults to True).
- **label_color (str)** -- (defaults to black).
- **fontsize** -- size of the fonts defaults to 'small'.
- **method** -- shape to be used in 'ellipse', 'square', 'rectangle', 'color', 'text', 'circle', 'number', 'pie'.
- **order_method** -- see `order()`. 
- **order_metric** -- see :meth:`order`.
- **cmap** -- a valid cmap from matplotlib or colormap package (e.g., 'jet', or 'copper'). Default is red/white/blue colors.
- **ax** -- a matplotlib axes.

The colorbar can be tuned with the parameters stored in `params`.

Here is an example. See notebook for other examples:

```python
import corrplot

c = corrplot.Corrplot(dataframe)
c.plot(cmap=('Orange', 'white', 'green'))
c.plot(method='circle')
c.plot(colorbar=False, shrink=.8, upper='circle')
```

**Heatmap, dendogram**

Heatmap and dendograms
class Clustermap(data_df, sample_groups_df=None, sample_groups_sel=[], sample_groups_palette=[], gene_groups_df=None, gene_groups_sel=[], gene_groups_palette=[], yticklabels="auto", **kwargs)

Heatmap and dendrograms based on seaborn Clustermap

```python
from sequana.viz.heatmap import Clustermap, get_clustermap_data
df, sample_groups_df, gene_groups_df = get_clustermap_data()
h = Clustermap(df, sample_groups_df=sample_groups_df, gene_groups_df=gene_groups_df)
h.plot()
```

Constructor

Parameters

- **data_df** -- a dataframe.
- **sample_groups_df** -- a dataframe with sample id as index (same as in data_df columns) and a group definition per column. Use to produce the x axis color groups.
- **sample_group_sel** -- a list of the columns to select from the sample_groups_df.
- **sample_groups_palette** -- the palette to use for sample color groups.
- **gene_groups_df** -- a dataframe with gene id as index (same as in data_df columns) and a group definition per column. Use to produce the y axis color groups.
- **gene_group_sel** -- a list of the columns to select from the gene_groups_df.
- **gene_groups_palette** -- the palette to use for gene color groups.
- **yticklabels** -- "auto" for classical heatmap behaviour, [] for no ticks or a pandas Series giving the mapping between the index (gene names in data_df) and the gene names to be used for the heatmap
- **kwargs** -- All other kwargs are passed to seaborn.Clustermap.

```
plot(cmap=None)
```

class Heatmap(data=None, row_method='complete', column_method='complete', row_metric='euclidean', column_metric='euclidean', cmap='yellow_black_blue', col_side_colors=None, row_side_colors=None, verbose=True)
Heatmap and dendograms of an input matrix

A heat map is an image representation of a matrix with a dendrogram added to the left side and to the top. Typically, reordering of the rows and columns according to some set of values (row or column means) within the restrictions imposed by the dendrogram is carried out.

```python
from sequana.viz import heatmap
df = heatmap.get_heatmap_df()
h = heatmap.Heatmap(df)
h.plot()
```

side colors can be added:

```python
h = viz.Heatmap(df, col_side_colors=['r', 'g', 'b', 'y', 'k']); h.category_column = category;
h.category_row = category
```

where category is a dictionary with keys as df.columns and values as category defined by you. The number of colors in col_side_colors and row_side_colors should match the number of category

**constructor**

**Parameters**

- **data** -- a dataframe or possibly a numpy matrix.
- **row_method** -- complete by default
- **column_method** -- complete by default. See linkage module for details
- **row_metric** -- euclidean by default
- **column_metric** -- euclidean by default
- **cmap** -- colormap. any matplotlib accepted or combo of colors as defined in colormap package (pypi)
• col_side_colors --
• row_side_colors --

property column_method
property column_metric

property df

property frame

plot(num=1, cmap=None, colorbar=True, vmax=None, vmin=None, colorbar_position='right', gradient_span='None', figsize=(12, 8), fontsize=None)

Using as input:

```python
df = pd.DataFrame({
'A':[1,0,1,1],
'B':[0.9,0.1,0.6,1],
'C':[0.5,2,0,1],
'D':[0.5,2,0,1])
```

we can plot the heatmap + dendogram as follows:

```python
h = Heatmap(df)
h.plot(vmin=0, vmax=1.1)
```

```python
from sequana.viz import heatmap
df = heatmap.get_heatmap_df()
h = heatmap.Heatmap(df)
h.category_column['A'] = 1
h.category_column['C'] = 1
h.category_column['D'] = 2
h.category_column['B'] = 2
h.plot()
```

property row_method

property row_metric

Hinton plot

Hinton plot

author  Thomas Cokelaer

hinton(df, fig=1, shrink=2, method='square', bgcolor='grey', cmap='gray_r', binarise_color=True)

Hinton plot (simplified version of correlation plot)

Parameters

• df -- the input data as a dataframe or list of items (list, array). See Corrplot for details.
• fig -- in which figure to plot the data
• shrink -- factor to increase/decrease sizes of the symbols
• method -- set the type of symbols for each coordinates. (default to square). See Corrplot for more details.
• **bgcolor** -- set the background and label colors as grey
• **cmap** -- gray color map used by default
• **binarise_color** -- use only two colors. One for positive values and one for negative values.

```python
from sequana.viz import hinton
df = np.random.rand(20, 20) - 0.5
hinton(df)
```

**Note:** Idea taken from a matplotlib recipes [http://matplotlib.org/examples/specialty_plots/hinton_demo.html](http://matplotlib.org/examples/specialty_plots/hinton_demo.html) but solely using the implementation within *Corrplot*

**Note:** Values must be between -1 and 1. No sanity check performed.

---

### 2D Histogram

2D histograms

**class** `Hist2D(x, y=None, verbose=False)`

2D histogram

```python
from numpy import random
from sequana.viz import hist2d
X = random.randn(10000)
Y = random.randn(10000)
h = hist2d.Hist2D(X, Y)
h.plot(bins=100, contour=True)
```
constructor

Parameters
- `x` -- an array for X values. See VizInput2D for details.
- `y` -- an array for Y values. See VizInput2D for details.

```
plot(bins=100, cmap='hot_r', fontsize=10, Nlevels=4, xlabel=None, ylabel=None, norm=None, range=None, normed=False, colorbar=True, contour=True, grid=True, **kwargs)
```

plots histogram of mean across replicates versus coefficient variation

Parameters
- `bins` (int) -- binning for the 2D histogram (either a float or list of 2 binning values).
- `cmap` -- a valid colormap (defaults to hot_r)
- `fontsize` -- fontsize for the labels
- `Nlevels` (int) -- must be more than 2
- `xlabel` (str) -- set the xlabel (overwrites content of the dataframe)
- `ylabel` (str) -- set the ylabel (overwrites content of the dataframe)
- `norm` -- set to 'log' to show the log10 of the values.
- `normed` -- normalise the data
- `range` -- as in pylab.Hist2D : a 2x2 shape [[-3,3],[-4,4]]
- `contour` -- show some contours (default to True)
- `grid` (bool) -- Show uerlying grid (defaults to True)

If the input is a data frame, the `xlabel` and `ylabel` will be populated with the column names of the data frame.

Image

Imshow utility

class Imshow(x, verbose=True)
Wrapper around the matplotlib.imshow function

Very similar to matplotlib but set interpolation to None, and aspect to automatic and accepts input as a data frame, in which case x and y labels are set automatically.

```python
import pandas as pd
data = dict([(letter, np.random.randn(10)) for letter in 'ABCDEFGHIJK'])
df = pd.DataFrame(data)

from sequana.viz import Imshow
im = Imshow(df)
im.plot()
```
**constructor**

**Parameters**

- **x** -- input dataframe (or numpy matrix/array). Must be squared.

```
plot(interpolation='None', aspect='auto', cmap='hot', tight_layout=True, colorbar=True, fontsize_x=None, fontsize_y=None, rotation_x=90, xticks_on=True, yticks_on=True, **kargs)
```

wrapper around imshow to plot a dataframe

**Parameters**

- **interpolation** -- set to None
- **aspect** -- set to 'auto'
- **cmap** -- colormap to be used.
- **tight_layout** --
- **colorbar** -- add a colobar (default to True)
- **fontsize_x** -- fontsize on xlabels
- **fontsize_y** -- fontsize on ylabels
- **rotation_x** -- rotate labels on xaxis
- **xticks_on** -- switch off the xticks and labels
- **yticks_on** -- switch off the yticks and labels
PCA

class PCA(data, colors={})

```python
from sequana.viz.pca import PCA
from sequana import sequana_data
import pandas as pd

data = sequana_data("test_pca.csv")
df = pd.read_csv(data)
df = df.set_index("Id")
p = PCA(df, colors={
    "A1": 'r', "A2": 'r', 'A3': 'r',
    "B1": 'b', "B2": 'b', 'B3': 'b'})
p.plot(n_components=2)
```

**constructor**

**Parameters**

- **data** -- a dataframe; Each column being a sample.
- **colors** -- a mapping of column/sample name a color

```python
plot(n_components=2, transform='log', switch_x=False, switch_y=False, switch_z=False, colors=None, max_features=500, show_plot=True, fontsize=10, adjust=True)
```

**Parameters**

- **n_components** -- at number starting at 2 or a value below 1 e.g. 0.95 means select automatically the number of components to capture 95% of the variance
- **transform** -- can be 'log' or 'anscombe', log is just log10. count with zeros, are set to 1

```python
plot_pca_vs_max_features(step=100, n_components=2, progress=True)
```

2.13. References (Viz) 179
ScatterPlot

Scatter plots

author Thomas Cokelaer

class ScatterHist(x, y=None, verbose=True)
Scatter plots and histograms

class ScatterHist

class ScatterHist

class ScatterHist

class ScatterHist

class ScatterHist

class ScatterHist

constructor

Parameters

- x -- if x is provided, it should be a dataframe with 2 columns. The first one will be used as your X data, and the second one as the Y data
- y --
- verbose --

plot(kargs_scatter={'c': 'b', 's': 20}, kargs_grids={}, kargs_histx={}, kargs_histy={}, scatter_position='bottom left', width=0.5, height=0.5, offset_x=0.1, offset_y=0.1, gap=0.06, facecolor='lightgrey', grid=True, show_labels=True, **kargs)
Scatter plot of set of 2 vectors and their histograms.

Parameters

- x -- a dataframe or a numpy matrix (2 vectors) or a list of 2 items, which can be a mix of list or numpy array. If size and/or color are found in the columns dataframe, those columns will be used in the scatter plot. kargs_scatter keys c and s will then be ignored. If a list of lists, x will be the first row and y the second row.
- y -- if x is a list or an array, then y must also be provided as a list or an array
- kargs_scatter -- a dictionary with pairs of key/value accepted by matplotlib.scatter function. Examples is a list of colors or a list of sizes as shown in the examples below.
- kargs_grid -- a dictionary with pairs of key/value accepted by the matplotlib.grid (applied on histogram and axis at the same time)
- kargs_histx -- a dictionary with pairs of key/value accepted by the matplotlib.histogram
- kargs_histy -- a dictionary with pairs of key/value accepted by the matplotlib.histogram
- kargs -- other optional parameters are hold, facecolor.
- scatter_position -- can be 'bottom right/bottom left/top left/top right'
- width -- width of the scatter plot (value between 0 and 1)
- height -- height of the scatter plot (value between 0 and 1)
- offset_x --
- offset_y --
- gap -- gap between the scatter and histogram plots.
- grid -- defaults to True

Returns the scatter, histogram1 and histogram2 axes.
```python
import pylab
import pandas as pd
X = pylab.randn(1000)
Y = pylab.randn(1000)
daframe = pd.DataFrame({'X': X, 'Y': Y})

from sequana.viz import ScatterHist
ScatterHist(df).plot()

from sequana.viz import ScatterHist
ScatterHist(x=[1, 2, 3, 4], y=[3, 5, 6, 4]).plot(
    kargs_scatter={'s': [200, 400, 600, 800],
                   'c': ['red', 'green', 'blue', 'yellow'],
                   'alpha': 0.5},
    kargs_histx={'color': 'red'},
    kargs_histy={'color': 'green'})

See also:

notebook
```

2.13. References (Viz)
Venn diagram

plot_venn(subsets, labels=None, title=None, ax=None, alpha=0.8, weighted=False, colors=('r', 'b', 'y'))

Plot venn diagramm according to number of groups.

Parameters subsets -- This parameter may be (1) a dict, providing sizes of three diagram regions. The regions are identified via two-letter binary codes ('10', '01', and '11'), hence a valid set could look like: {'10': 10, '01': 20, '11': 40}. Unmentioned codes are considered to map to 0. (2) a list (or a tuple) with three numbers, denoting the sizes of the regions in the following order: (10, 01, 11) and (3) a list containing the subsets of values.

The subsets can be a list (or a tuple) containing two set objects. For instance:

```
from sequana.viz.venn import plot_venn
A = set([1,2,3,4,5,6,7,8,9])
B = set([7,8,9,10,11])
plot_venn((A, B), labels=('A', 'B'))
```

This is the unweighted version by default meaning all circles have the same size. If you prefer to have circle scaled to the size of the sets, add the relevant parameter as follows:

```
from sequana.viz.venn import plot_venn
A = set([1,2,3,4,5,6,7,8,9])
B = set([7,8,9,10,11])
plot_venn((A, B), labels=('A', 'B'), weighted=True)
```

Similarly for 3 sets, a Venn diagram can be represented as follows. Note here that we also use the title parameter:
```python
from sequana.viz.venn import plot_venn

A = set([1,2,3,4,5,6,7,8,9])
B = set([ 4,5,6,7,8,9,10,11,12,13])
C = set([ 3,4,5,6,7,8,9])
plot_venn((A, B, C), labels=('A', 'B', 'C'), title='my Venn3 diagram')
```

Input can be a list/tuple of 2 or 3 sets as described above.

**Volcano plots**

Volcano plot

```python
class Volcano(data=None, log2fc_col='log2FoldChange', pvalues_col='padj', annot_col='', color='auto',
              pvalue_threshold=1.3010299956639813, log2fc_threshold=1)
```
import pandas as pd

data = pd.DataFrame([fc, pvalue])
data = data.T

data.columns = ['log2FoldChange', 'padj']

from sequana.viz import Volcano

v = Volcano(data)
v.plot()

---

constructor

Parameters

- data (DataFrame) -- Pandas DataFrame with rnadiff results.
- log2fc_col -- Name of the column with log2 Fold changes.
- pvalues_col -- Name of the column with adjusted pvalues.
- annot_col -- Name of the column with genes names for plot annotation.
- color -- for color choice
- pvalue_threshold -- Adjusted pvalue threshold to use for coloring/annotation.
- log2fc_threshold -- Log2 Fold Change threshold to use for coloring/annotation.

annotate(**kwargs)
plot(size=10, alpha=0.7, marker='o', fontsize=16, xlabel='fold change', logy=False, threshold_lines={'color': 'black', 'ls': '--', 'width': 0.5}, ylabel='p-value', add_broken_axes=False, broken_axes={'ylims': ((0, 10), (50, 100))})

**Parameters**

- **size** -- size of the markers
- **alpha** -- transparency of the marker
- **fontsize** --
- **xlabel** --
- **ylabel** --
- **center** -- If centering the x axis

### 2.14 References (stats)

#### 2.14.1 Statistical tools

Statistical tools

**evenness(data)**

Return Evenness of the coverage

**Reference** Konrad Oexle, Journal of Human Genetics 2016, Evaluation of the evenness score in NGS. work before or after normalisation but lead to different results.

\[ C = \text{mean}(X) \]  
\[ D^2 = X[X < C] \]  
\[ N = \text{len}(X) \]  
\[ n = \text{len}(D) \]  
\[ E = 1 - \frac{(n - \text{sum}(D^2)/C)}{N} \]

**moving_average(data, n)**

Compute moving average

**Parameters** **n** -- window’s size (odd or even).

```
>>> from sequana.stats import moving_average as ma
>>> ma([1,1,1,1,3,3,3,3], 4)
array([ 1. , 1.5, 2. , 2.5, 3. ])
```

**Note:** the final vector does not have the same size as the input vector.

**runmean(data, n)**

Akaike and other criteria
AIC($L, k, logL=False$)
Return Akaike information criterion (AIC)

**Parameters**

- $L$ (float) -- maximised value of the likelihood function
- $k$ (int) -- number of parameters
- $logL$ (bool) -- L is the log likelihood.

Suppose that we have a statistical model of some data, from which we computed its likelihood function and let $k$ be the number of parameters in the model (i.e. degrees of freedom). Then the AIC value is:

$$AIC = 2k - 2 \ln(L)$$

Given a set of candidate models for the data, the preferred model is the one with the minimum AIC value. Hence AIC rewards goodness of fit (as assessed by the likelihood function), but it also includes a penalty that is an increasing function of the number of estimated parameters. The penalty discourages overfitting.

Suppose that there are R candidate models AIC1, AIC2, AIC3, AICR. Let AICmin be the minimum of those values. Then, $\exp((AICmin - AIC_i)/2)$ can be interpreted as the relative probability that the ith model minimizes the (estimated) information loss.

Suppose that there are three candidate models, whose AIC values are 100, 102, and 110. Then the second model is $\exp((100 - 102)/2) = 0.368$ times as probable as the first model to minimize the information loss. Similarly, the third model is $\exp((100-110)/2) = 0.007$ times as probable as the first model, which can therefore be discarded.

With the remaining two models, we can (1) gather more data, (2) conclude that the data is insufficient to support selecting one model from among the first two (3) take a weighted average of the first two models, with weights 1 and 0.368.

The quantity $\exp((AIC_{min}-AIC_i)/2)$ is the relative likelihood of model i.

If all the models in the candidate set have the same number of parameters, then using AIC might at first appear to be very similar to using the likelihood-ratio test. There are, however, important distinctions. In particular, the likelihood-ratio test is valid only for nested models, whereas AIC (and AICc) has no such restriction.


AICc($L, k, n, logL=False$)
AICc criteria

**Parameters**

- $L$ (float) -- maximised value of the likelihood function
- $k$ (int) -- number of parameters
- $n$ (int) -- sample size
- $logL$ (bool) -- L is the log likelihood.

AIC with a correction for finite sample sizes. The formula for AICc depends upon the statistical model. Assuming that the model is univariate, linear, and has normally-distributed residuals (conditional upon regressors), the formula for AICc is as follows:

AICc is essentially AIC with a greater penalty for extra parameters. Using AIC, instead of AICc, when n is not many times larger than $k^2$, increases the probability of selecting models that have too many parameters, i.e. of overfitting. The probability of AIC overfitting can be substantial, in some cases.
**BIC** \(L, k, n, \log L=False\)

Bayesian information criterion

**Parameters**

- \(L\) (float) -- maximised value of the likelihood function
- \(k\) (int) -- number of parameters
- \(n\) (int) -- sample size
- \(\log L\) (bool) -- \(L\) is the log likelihood.

Given any two estimated models, the model with the lower value of BIC is the one to be preferred.

**class** `EM(data, model=None, max_iter=100)`

Expectation maximization class to estimate parameters of GMM

```python
from sequana import mixture
from pylab import normal

data = [normal(0,1) for x in range(7000)] + [normal(3,1) for x in range(3000)]
em = mixture.EM(data)
em.estimate(k=2)
em.plot()
```

**constructor**

**Parameters**

- `data` --
- `model` -- not used. Model is the `GaussianMixtureModel` but could be other model.
- `max_iter` (int) -- max iteration for the minimization

**estimate** (`guess=None, k=2`)

**Parameters**

- `guess` (list) -- a list to provide the initial guess. Order is mu1, sigma1, pi1, mu2, ...
- `k` (int) -- number of models to be used.
plot(model_parameters=None, **kwargs)

Take a list of dictionaries with models parameters to plot predicted models. If user doesn't provide parameters, the standard plot function from fitting is used.

Example: model_parameters=[{"mu": 5, "sigma": 0.5, "pi": 1}]

class Fitting(data, k=2, method='Nelder-Mead')

Base class for EM and GaussianMixtureFitting

constructor

Parameters

• data (list) --
• k (int) -- number of GMM to use
• method (str) -- minimization method to be used (one of scipy optimise module)

get_guess()

Random guess to initialise optimisation

property k

property model

plot(normed=True, N=1000, Xmin=None, Xmax=None, bins=50, color='red', lw=2, hist_kw={'color': '#5F9EA0', 'edgecolor': 'k'}, ax=None)

class GaussianMixtureFitting(data, k=2, method='Nelder-Mead')

GaussianMixtureFitting using scipy minization

from sequana import mixture
from pylab import normal
data = [normal(0,1) for x in range(700)] + [normal(3,1) for x in range(300)]
mf = mixture.GaussianMixtureFitting(data)
mf.estimate(k=2)
mf.plot()

Here we use the function minimize() from scipy.optimization. The list of (currently) available minimization methods is 'Nelder-Mead' (simplex), 'Powell', 'CG', 'BFGS', 'Newton-CG', 'Anneal', 'L-BFGS-B' (like BFGS but bounded), 'TNC', 'COBYLA', 'SLSQPG'.

estimate(guess=None, k=None, maxfev=20000.0, maxiter=1000.0, bounds=None)

guess is a list of parameters as expected by the model

guess = {'mus':[1,2], 'sigmas': [0.5, 0.5], 'pis': [0.3, 0.7] }

property method

class GaussianMixtureModel(k=2)

Gaussian Mixture Model

log_likelihood(params, sample)
\texttt{pdf(x, params, normalise=True)}

Expected parameters are

params is a list of gaussian distribution ordered as mu, sigma, pi, mu2, sigma2, pi2, ...

class MetropolisHasting

![Simulated (blue) and target (red) distributions](image)

property Xtarget

property Ytarget

\texttt{check(bins=60)}

\texttt{diagnostics(bins=60, clear=True)}

\texttt{simulate(n=100000, burning=20000, step=None, x0=None)}

Simple VST transformation

class VST

```python
v = VST()
v.anscombe(X)
```

**static anscombe**

Compute the anscombe variance stabilizing transform.

**Parameters**

- **x** -- noisy Poisson-distributed data

**Returns**

- data with variance approximately equal to 1.


For Poisson distribution, the mean and variance are not independent. The anscombe transform aims at transforming the data so that the variance is about 1 for large enough mean; For mean zero, the variance is still zero. So, it transform Poisson data to approximately Gaussian data with mean $\sqrt{x + \frac{3}{8} - \frac{1}{(4m^{1/2})}}$

**static generalized_anscombe**

Compute the generalized anscombe variance stabilizing transform

Data should be a mixture of poisson and gaussian noise.

The input signal $z$ is assumed to follow the Poisson-Gaussian noise model:

$$x = \text{gain} \ast p + n$$

where gain is the camera gain and mu and sigma are the read noise mean and standard deviation. X should contain only positive values. Negative values are ignored. Biased for low counts

**static inverse_anscombe**

Compute the inverse transform


### 2.15 FAQs

#### 2.15.1 Conda related

Create a conda environment on IP cluster:

```bash
module load conda
conda create --name py35 python=3.5
source condaenvs/py35/bin/activate py35
```

add channel from where to download packages:

```bash
conda config --add channels r bioconda
conda install sequana
```
2.15.2 What are the dependencies

There are two kinds of dependencies. First, the Python libraries such as matplotlib or Pandas. Second, the external tools such as BWA (alignment) or Kraken (taxonomy). The first kind of tools can be installed using Anaconda and the default conda channel. For instance:

```
conda install pandas
```

The second kind of tools can also be installed using another conda channel called `bioconda`. For instance:

```
conda install bwa
```

The full list of dependencies will be maintained in the `Installation` section but those two lines should be sufficient to install most of the dependencies with `conda`:

```bash
conda install --file https://raw.githubusercontent.com/sequana/sequana/master/requirements.txt
conda install --file https://raw.githubusercontent.com/sequana/sequana/master/requirements_pipelines.txt
```

Additional tools such as prokka, busco, canu and future heavy software will be maintained in this specific requirements for now:

```bash
conda install --file https://raw.githubusercontent.com/sequana/sequana/master/requirements_pipelines_extra.txt
```

2.15.3 Installation issues

As explained in the previous section, most of the dependencies can be installed via Conda. If not, pip is recommended. Yet there are still a few dependencies that need manual installation.

**quast**

http://quast.bioinf.spbau.ru/manual.html#sec1

```
wget https://downloads.sourceforge.net/project/quast/quast/quast-4.2.tar.gz
tar -xzf quast-4.2.tar.gz
cd quast-4.2
```

Alternatively, get the source code from their GitHub (takes a while):

```
git clone https://github.com/ablab/quast
cd quast
python setup.py install
```
graphviz

graphviz provides an executable called dot. If you type dot in a shell and get this error message:

```
Warning: Could not load ...
```

This may be solved by re-installation graphviz using the main anaconda channel (instead of bioconda):

```
conda install --override-channels -c anaconda graphviz=2.38.0
```

Update April 2017 replace anaconda with conda-forge

matplotlib

If you get errors related to the X connection, you may need to change the backend of matplotlib. To do so, go in your home directory and in this directory

```
cd /home/user/.config/matplotlib/
```

Check if the file `matplotlibrc` exits, if not, type:

```
echo "backend: Agg" > matplotlibrc
```

or edit the file and make sure the line starting with "backend" uses the Agg backend:

```
backend: Agg
```

Save, exit the shell, start a new shell.

pysam / samtools / bzip2

We have experienced few issues with pysam and samtools. Here are some solutions.

```
from pysam.libchtslib import *
...ImportError: libhts.so.1: cannot open shared object file: No such file or directory
```

This may be solved by removing conda installation and using pip instead:

```
conda remove pysam
pip install pysam
```

Another error know for pysam version 0.11.2.2 raises this error:

```
ImportError: libb2.so.1.0: cannot open shared object file: No such file or directory
```

Downgrading to version 0.11.2.1 and upgrading to working version solves the problem:

```
conda install pysam=0.11.2.1
```

but one reason was also related to the order of the channel in the .condarc file. You may get bzip2 from the default channel and not from conda-forge (reference: https://github.com/bioconda/bioconda-recipes/issues/5188)
conda install --override-channels -c conda-forge bzip2

pysam may not compile due to a missing dependency on lzma. Under fedora, type:

```
yum install liblzma liblzma-devel
```

### qt and pyqt

#### Qt Version

With PyQt 5.12.3 and python3.7, we got lots of errors:

```
SystemError: <built-in function connectSlotsByName> returned a result with an error set
```

This seems to be a PyQt bug according to several github projets based on pyqt. It may be fixed a version above. Dowgrading e.g. to pyqt 5.9.2 does not solve the problem.

#### Qt compatibility across platform

```
from PyQt5.QtWebKitWidgets import QWebView
...ImportError: libQt5WebKitWidgets.so.5: cannot open shared object file: No such file or directory
```

This may be solved by re-installation qt using the main anaconda channel (instead of bioconda):

```
conda install --override-channels -c anaconda qt
```

and possibly:

```
pip install PyQtWebEngine
```

If we believe this issue: https://github.com/conda-forge/pyqt-feedstock/issues/19

#### libselinux

If you get this error (using conda install sequana):

```
ImportError: libselinux.so.1: cannot open shared object file: No such file or directory
```

it looks like you need to install libselinux on your environment as reported here.
pytz installation

If you get this error:

```
ImportError: C extension: No module named 'pytz.tzinfo' not built. If you
want to import pandas from the source directory, you may need to run 'python
setup.py build_ext --inplace --force' to build the C extensions first.
```

try this:

```
pip uninstall pytz
pip install --pre pytz
```

reference: https://github.com/sequana/sequana/issues/499

2.15.4 Expected input format

Most of the pipelines and standalone expect FastQ files with the extension fastq.gz meaning that files are gzipped. Besides, the filename convention is as follows:

```
PREFIX_R1_.fastq.gz
```

that is _R1_ and _R2_ indicates the paired or single-ended files and the PREFIX is used to create directories or reports; it must be present.

New in version 0.2: more flexible tags are now possible in sequana pipelines and sequanix using e.g. _R[12] in the input_readtag in the configuration file of the pipelines.

2.15.5 Sequanix related

For question related to Sequanix, we have a dedicated section in FAQS.

2.15.6 QXcbConnection issue

If you get this error:

```
QXcbConnection: Could not connect to display localhost:10.0
```

this is an issue with your Qt backend. You need to change it to Agg.

2.15.7 Variant Calling pipeline

If snpeff fails with this type of errors:

```
java.lang.RuntimeException: Error reading file 'null'
java.lang.RuntimeException: Cannot find sequence for 'LN831026.gbk'
```

this may be because your genbank does not contain the sequences.

Another type of errors is that the sequence and genbank are not synchrone. We would recommend to use the code here to download the Fasta and genbank:

2.15.8 Quality Control pipeline

Please see the tutorial, user guide or pipelines section and look for the quality control.
Then, if you do not find your solution, please open an issue on github: https://github.com/sequana/sequana/issues

2.15.9 Singularity

If you use the singularity container and get this kind of error:

```
singularity shell sequana-sequana-master.img
ERROR : Base home directory does not exist within the container: /pasteur
ABORT : Retval = 255
```

it means the container does not know about the Base home directory.

If you have sudo access, add the missing path as follows:

```
sudo singularity shell --writable sequana-sequana-master.img
mkdir /pasteur
exit
```

If you do not have sudo permissions, copy the image on a computer where you have such permission, use the same code as above and copy back the new image on the computer where you had the issue.

Finally, try to use the container again using this code:

```
singularity shell sequana-sequana-master.img
```

2.15.10 I got a error "main thread is not in the main loop"

```
Traceback (most recent call last):
  File ".../lib/python3.5/tkinter/__init__.py", line 627, in after_cancel
    data = self.tk.call('after', 'info', id)
RuntimeError: main thread is not in main loop
```

This is related to the backend used by matplotlib. This can be ignored. We do not have any solution for now, except finding an alternated backend for matplotlib. This can be done using a special file called matplotlibrc with this content:

```
backend: tkagg
```

where you can replace tkagg with e.g. qt5agg
2.15.11 Installation issue on Mac

On a MacOSx conda environment (PYthon3.9), I could not build datrie with this kind of error message:

```
error: command 'llvm-ar' failed: No such file or directory
ERROR: Failed building wheel for datrie
Failed to build datrie
Failed to build datrie
ERROR: Could not build wheels for datrie, which is required to install pyproject.toml-based projects
```

The solution was to set the AR variable:

```
export AR=/usr/bin/ar
```

2.16 Changelog

2.16.1 0.12.3

- CHANGES:
  - allow feature counts to scan the new layout. Back compatible so nothing to change in your code.

2.16.2 0.12.2

- CHANGES
  - Update itol module to use APIkey (change of the web services)

2.16.3 0.12.1

- CHANGES
  - fix blocking bug in laa multiqc module
  - Updates of the rnadiff subcommand

2.16.4 0.12.0

- CHANGES
  - databases.ENADownload class removed (the ftp used are deprecated and not maintained by EBI)
  - kraken refactoring: kraken1 support removed. kraken_builder.py module not required anymore. Old code still interesting has been moved to databases.py The kraken.py module was split into analysis.py and multikraken.py into a new sub directory called kraken. This should be transparent for the user and developer. See PR
  - All tools related to pipelines have be move to an external Python package (sequana_pipetools, on pypi )
  - All tools related to sphinx extension have be move to an external Python package (sequana_sphinext, on pypi)
– Rules are still in used but will be replaced little by little by an external Python package (sequana-wrappers, on github)
– github action now included for a better CI
– Lots of cleanup !

2.16.5 0.11.1

• hotfix on rule bowtie2_index

2.16.6 0.11

• new tools and functionalities for the rnaseq pipeline
• add binaries attribute in rules handled by snakertools

2.16.7 0.10.0

MAJOR release merging the rnadiff branch into the dev branch so as to finalise the way differential analysis are handle in Sequana with respect to RNA-seq analysis
Module concerned: rnadiff, compare, featurecounts and enrichment

• CHANGES:
  – removed expdesign module. Improved iem module to cope with old formats. From now on, we will use IEM formats only for the illumina sample sheets. No more home-made formats will be supported.
• NEWS:
  – module kegg and command in sequana (sequana taxonomy) to search for a given name in all KEGG organism so as to easily retrieve the taxon and organism name

2.16.8 0.9.8

• BUGS:
  – fastqc.FastQC class can now handle empty samples (no reads... yes it may happen)
  – fastq random read selection corrected to handle paired data
• IMPROVEMENTS:
  – CLI sequana summary can parse GFF and print summary information
  – add custom section for Makefiles in pipelines
  – add an unlock.sh for all pipelines
  – featurecounts: allow input to use the --extraAttributes option; more tests
• NEWS:
  – gff 1. add save_gff_filtered method to select interesting features, 2. add gff2gtf conversion
  – fasta module has GC content method
  – new rule macs3_dynamic
2.16.9 0.9.7

- rnadiff HTML report can now show the locus_tag in volcano plot
- rules star_index uses floor instead of round for small genomes to agree with star index recommendation
- add MultiFeatureCounts() class to read several feature counts

2.16.10 0.9.6

- **BUGS:**
  - Fix star_index rule (regression bug) reported in https://github.com/sequana/sequana_rnaseq/issues/7
- **IMPROVEMENTS/CHANGES:**
  - add plot() method in Repeats class
  - featurecounts: sort the output dataframe columns
  - sequana CLI: summary/enrichment sub command allows to switch between log/linear scale + option to compute the GO levels and empty plots are now shown

2.16.11 0.9.5

- rules fastqc: allows handling of bam files. remove sanity checks on empty files.
- rules falco: new rules added (similar to fastqc)
- module snaketools: PipelineManagerGeneric set paired field to "undefined" since this is suppose to deal with bam, or other non-fastq files
- module fastqc now plots the pacbio phred score on the full y-axis range
- for developers: In SequanaManager teardown, option to skip check_fastq_files

2.16.12 0.9.4

- faster gff module to be used by sequana_rnaseq>=0.9.19

2.16.13 0.9.3

**general**

- finalise the rnadiff HTML report with plotly visualisation
- finalise the enrichmentf HTML report
- template for rnadiff (rna differential analysis)
- method in FastQ to extract some specific reads
- add new application in main sequana tool: - biomart: a tool to produce a CSV for a given dataset!mart/set of attributes - summary: a tool to produce HTML for bam, fasta/fastq/bam, rnadiff results, enrichment for a rnadiff result - fastq: can now merge several files - gtf_fixer that replaces the script sequana_gtf_fixer
- add quick_fix for sample sheet errors in iem.IEM class (used in sequana_demultiplex pipeline)
- Fix featurecounts package to include tolerance parameter
• Fix bug for category in Dendogram

for developers

• rules: star_index now sets automatically the IndexNbases parameter
• removed sequana_summary from standalone now in the general sequana CLI
• improved find_motif module with plots, pep8, better API
• js: reverse papaparse

2.16.14 0.9.2

• add salmon module for the rnaseq pipeline
• fix bug in multiqc plugin for bamtools used in sequana_mapper
• some rules updates: (1) add_read_group has now the option "VALIDATION_STRINGENCY=SILENT" that seem to be required in newer version of picard. surprising that this error did not appear earlier since the issue seems old (https://github.com/Cibiv/NextGenMap/issues/3) (2) mark_duplicates and features_counts rules needed o change some parameters, now keywords in snakemake
• add new viz module called dendogram
• add new report module for rnadiff.
• add new report module for KeggPathway and Panther
• Add solution to convert ensembl ID to gene name fr the KeggPathwayEnrichment
• Sequana main script has now a 'summary' command. Integrated the RNADiff and BAMQC modules for now.

2.16.15 0.9.1

• Fix html syntax in the templates
• Fix demultiplex.barplot_samples layout
• Created a sequana main script with currentl one command, fastq, which replaces sequana_fastq_head, sequana_fastq_count, etc

2.16.16 0.9.0

• Fix requirements (missing stattmodels)
• Fix bug kraken sequential when no reads are classified
• Fix FileFactory to make it much faster (speed up pipeline initialisation). This should also fix a recurrent issue with read tag (see https://github.com/sequana/sequana/issues/526
• NEWS:
  – implemented rnadiff module
  – implemented go term enrichment
  – implemented feature counts
  – implemented plot.viz.venn
– implemented compare module (for rnadiff comparison)

2.16.17 0.8.6

• CHANGES:
  – module kraken: better label in plot2 method to avoid long label to be cut
  – module compare: fix label in comparative plots
  – module modules_report/bases now include title in the embedded image

2.16.18 0.8.5

• CHANGES:
  – coverage multiqc plugin to remove unique chromomosome name from the sample name, if it is unique
  – bamtools_stats multiqc plugin to remove unique chromomosome name from the sample name, if it is unique
  – fix regression bug in report summary page for the taxonomy tools

• NEWS:
  – Contigs now acceps Spades fasta files
  – added a new viz module for PCA plotting
  – added a new stats module for data normalisation
  – added a new enrichment module to include KeggPathway enrichment

2.16.19 0.8.4

• MultiKrakenResults set nan tp zeros

• fasta module: new filter() method to keep or exclude some entries

• CHANGES:
  – pipeline_common: PipelineManager renamed into SequanaManager to avoid same name as in snake-tools.PipelineManager
  – snakertools: PipelineManagerGeneric is now the base class for PipelineManager. factorise common methods into PipelineManagerGeneric
  – SnakeMakeStats now skip the figure if stats is empty

• NEWS:
  – snakertools: new PipelineManagerDirectory class (used by sequana_demultiplex) when no input files are used, but only an input directory. we can now import exiting project to fill the config file automatically
  – module fastqc to plot static images of fastqc contents. The parsing method was adapted from multiqc.module.fastqc itself as a quick hack to incorporate in the sequana_fastqc pipeline
2.16.20 0.8.3

- Pipeline related:
  - add plot_stats function in PipelineManager
  - add setup/teardown function in PipelineManager
- CHANGES:
  - new dependency (bx) to include new features in bamtools module (infer strandness, insert size etc)
  - sequana_taxonomy now handles both kraken1 and kraken2 databases the taxonomy databases is now downloaded from ncbi directly and build within sequana. The standalone can download the taxonomy data itself with --download-taxonomy option. New pipeline available on github/sequana/sequana_taxonomy
  - rename KrakenHierarchical into KrakenSequential
  - refactoring of kraken.py and kraken_builder to include kraken2
  - taxonomy module can now aggregate several taxonomic databases. We now retrieve the input data from NCBI instead of EBI to be up-to-date whenever we want (e.g. daily basis)
- BUGs:
  - add package packaging in requirements (for travis and pip installation)
- NEWs
  - 7 new modules in sequana.viz dedicated to visualisation, in particular dendogram, corrplot, hist2D from biokit project (merging with sequana) together with a bunch of notebooks
  - NEW bed module to include a BED class to read 12-column BED files.
  - gff3 module: a new method to create utility files for the rnaseq pipeline
  - New rule for the rnaseq pipeline: salmon
  - kraken2 database can now be used within sequana_taxonomy
- General cleaning: lane_merging tools with tests, deprecated warnings
- Move from 70 to 85% of testing coverage

2.16.21 0.8.2

- a quick fix in the requirements to build proper recipes on bioconda, now available on bioconda channel (1 April 2020).

2.16.22 0.8.1

- add missing dependency packaging in requirements.txt
- add new sphinx extension to include external pipeline documentation
- for pipelines, we created a common epilog and prolog statement and a init_pipeline to insert before parsing the options. This allows to have the --deps argument to print all dependencies of a pipeline
2.16.23 0.8.0 (24 March 2020)

This is an stable release made to includes lots of new features and pipelines and bug fixes made.

We removed all pipelines from sequana. Pipelines have now their own repositories on github to ease the developpement of sequana and those pipelines. The rationale being that we do not need to update sequana when a pipeline changes and a pipeline can have its own biocontainer and life cycle. We kept just one for testing purposes (compressor). Other pipelines have now their own repositories:

- sequana_coverage
- sequana_demultiplex
- sequana_fastqc
- sequana_revcomp
- sequana_rnaseq
- sequana_variant_calling
- sequana_denovo
- sequana_pacbio_amplicon
- etc

New pipelines have been added such as sequana_downsampling and sequana_mapper.

- NEWS:
  - snpeff now includes GFF input file input as an option (in addition to the GBK)
  - new module trf for tandem repeat finder output.
  - new scripts:
    * sequana_start_pipelines to initiate a new pipeline from the cookiecutter recipes
    * sequana_gtf_fixer to help fixing GTF issues
  - new module gff3 to read GFF files
  - Module can now encapsulate logos
  - Module version implemented

- BUG:
  - snpeff_add_locus_tag: if contig name and length in GFF header not in the same order as in the fasta, a new fasta with wrong header was created. This caused trouble in the variant_calling pipeline
  - kraken: Fix kraken plot (matplotlib version) when 100% of the reads are classified
  - Header of igvtools count output may vary. Make the consensus.get_bases more robust to automatically identify number of lines to skip.
  - Fix the kraken multiqc report
  - Fix bug in gui/browser to fix import of QWebPage on travis
  - bowtie2 dynamic rule now uses templating correctly (RNASeq pipeline)
  - Fix issue in snakertools for input_readtag set to _[12] for paired data The paired attribute was wrongly set to unpaired. Besides, we make it more robust for those who tag their paired data with _1 and _2 instead of _R1/_R2_
Repeats: for multi fasta with similar header, we were expecting the chrom name to be unique but underlying tool uses regular expression. So, this was buggy when chrom name were starting with same string. e.g chr1 and chr11.

multicq section of sequana_coverage: duplicate chrom names across multiple samples were shown as a single entry in the report.

draft version of multicq for sequana_quality_control now available

• MAJOR CHANGES/FIXES:

  – The main script ‘sequana’ is redundant with the new framework of pipelines. It has been removed in this version
  – sequana_coverage now handles low coverage correctly in the HTML reports. Fix the ylims of the coverage plot for low coverage.
  – cutadapt rules was failing due to a stricter optional/positional argument handling. Fixed the rule accordingly.
  – sequana_lane_merging is now ready for production. changes made: copy of the script in the local directory, not the data directory.
  – New pipeline_common module to be used by all pipelines

• MINOR CHANGES/FIXES

  – snaketools:
    * pipelines discovery updated in ModuleFinderSingleton. Finally fixed the lost of comments in the config when saved.
    * Removed onweb() method.
    * Fixed the loss of comments when saving yaml file after an update of the key/value.
    * remove check_sequana_fields.
    * more tests and cleanup
  – demultiplex: fix a Pandas deprecated warning (add sort argument in pd.concat)
  – python dependencies not in conda are not harcoded inside the setup.py (itolapi). add cython into the list of requirements.
  – Fix deprecated bamCoverage rule to use newest deeptools version.
  – The check_config_with_schema function now performs the validation correctly
  – Fix stdout of the fastqc, unpigz, bowtie1 and bowtie2 rules
  – Atropos 2.0 changed its API. fastq module compat with atropos 1.0 and 2.0
2.16.24 0.7.2

• NEWS:
  – New script: sequana_fastq_summary included in fastqc pipeline
  – New script: sequana_substractor to remove reads that mapped against a reference(s)
  – added a new module to upload/export phylogenetic tree on itol website. Used in the laa pipeline
  – added backspace2fusion code to merge lanes in Illumina raw data
  – added new pipeline called fastqc to simply run fastqc + multiqc in parallel
  – added laa pacbio pipeline
  – multiqc modules: bamtools_stats and kraken module for the laa pipeline
  – added test file and test for SIRVRerence class (partial fix of issue #504)
  – added Makefile class in snaketools to help building pipeline
  – added MultiKrakenResults class
  – sequanix and snaketools now handle the presence of a multiqc_config file in the pipeline module
  – add laa multiqc

• BUGS:
  – in quality_control when using the design file in cutadapt rule
  – Fix multiqc report for pacbio_qc pipeline

• CHANGES:
  – adapters added: TruSeqCD, TruSeqUD, etc
  – adapters removed: rubicon
  – remove clean_ngs rule and code related to this software, not used in sequana

• CHANGES for developers:
  – adapters are now named NAME_fwd.fa instead of adapters_NAME_fwd. This should not affect the user interface. Also, the index sequence stored in the adapter files are now identical in the forward/reverse/revcomp versions to simplify the code. We also added a script in ./resources/data/adapters to create the rev and revcomp version automatically.
  – add missing xlrd dependencies in requirements

2.16.25 0.7.1

• NEWS:
  – added metropolis hastings module
  – added a sniffer module for BAM/SAM/CRAM
  – added a SMA/CRAM reader

• CHANGES:
  – refactoring of bamtools. added SAM and CRAM classes. remove the plot_acgt_content method. Instead of inheriting from pysam.Alignement, we store the data as an attribute.

• FIXES:
– cutadapt rules and expdesign can now handle sample names with several underscores
– Issue 515: sequanix should now be able to handle list in YAML files
– Issues 520: level info in sequanix was always set to INFO at start time
– Issue 519: fix issues in sequanix due to different API in new ruamel.yaml version
– Issue #522: fix bam_splitter tool

2.16.26 0.7.0

• BUGS:
  – add /1 and /2 in quality control pipeline https://github.com/sequana/sequana/issues/508
  – Fix test failure due to freebayes version 1 and 1.2 https://github.com/sequana/sequana/issues/512
  – Fix reading of SampleSheet for MiSeq: https://github.com/sequana/sequana/issues/511
  – Add Exp Design checked in quality control pipeline: https://github.com/sequana/sequana/issues/500

• CHANGES:
  – sequana_vcf_filter: finalised version with INDEL removal, filters on DP4 and AF1 fields
  – rename PacbioBAM into PacbioSubreads

2.16.27 0.6.X

0.6.5

• CHANGES:
  – sequana_coverage. Major refactoring of bedtools module to handle large data sets (human), and provide ability to focus on CNVs using an additional naive clustering (merge_rois_into_cnvs method) and binning. We can also analyse data chunk by chunk (to avoid filling the memory). added a plot_rois function
  – sequana_coverage standalone: add the --chunksize, --cnv-clustering and --binning options.

• NEWS:
  – add cnvnator class
  – coverage pipeline added in the pipelines

• BUGS:
  – Fix silent warning (regex) in snpeff module
  – double indexing adapters issue for Nextera fixed: https://github.com/sequana/sequana/issues/501
0.6.4

• **BUGS:**
  – Fix issue https://github.com/sequana/sequana/issues/380 is_sorted property of the BAM class.
  – Fix --no-report option in sequana_coverage and add --clustering (double threshold option)
  – pacbio_qc pipeline is now able to also read old pacbio format

• **NEWS:**
  – SARTools rule added and used in the RNAseq pipeline
  – add summary module to store summary in json formats.
  – simple vcf_filter standalone

• **CHANGES:**
  – pin kraken version to 1.1 (newest on bioconda)
  – MAJOR REFACTORING of bedtools and sequana_coverage standalone. In particular, change default window size to 20,001 or a fifth of genome length (for small genome); speed up code; add plot_roi function, uses multiqc for summary page; add log2 ratio column. See https://github.com/sequana/sequana/issues/495 for details. Scan large files by chunk. Add a snakemake that can be used in sequanix.
  – remove the sequana_report standalone, which was not finalised and won’t be used in the future. We will use multiqc instead.

0.6.3.post1

• a bug fix in the sequanix GUI and singularity. A statement related to the ruamel.yaml package causes trouble if version is not 0.15. A temporary fix consisted in adding a try/except (the statement is just a warning.filter and has no impact on analysis)

0.6.3

• **BUGS:**
  – Fix bug in the copy of the fastqc data sets in the quality control
  – atropos bug in the reports (not full). Bug reported to atropos github. https://github.com/jdidion/atropos/issues/57. Need to use version 1.1.16
  – kraken report table were not sorted by percentage (as expected). Also, if the case of poor databases with few entries, the output may contain lots of classified sequences with Taxon 1, which was not reported correctly in the krona plot.

• **NEWS:**
  – example of a schema.yaml implemented for the quality control.
  – sequanix: reads schema.yaml automatically for sequana pipelines and can import one for generic cases. An option in the preference was added to switch on/off the validation of the config file with this schema. Can also import schema file for the generic case.

• **CHANGES:**
  – Taxonomy file is downloaded for Kraken only when Kraken is used, not in the main __init__ file anymore.
0.6.2

- **BUGS:**
  - Fix missing N_final column in table of the quality_control multi-summary page
  - Remove phix174.fa requirements in RNAseq pipeline config file
  - Fix path starting with tilde ([https://github.com/sequana/sequana/issues/486](https://github.com/sequana/sequana/issues/486))

- **NEWS:**
  - add isoseq Class
  - add vcf_filter module back to help in filtering VCF files created with mpileup for instance
  - add sequana_vcf_filter standalone
  - add cigar module to help deciphering CIGAR strings

0.6.1

- **BUGS:**
  - Fix empty dependency list in HTML report if sequana installed with conda

0.6.0

- **BUGS:**
  - add missing file for the RNAseq pipeline in the setup.py
  - Fix RTD building
  - Fix reag_tag filtering [https://github.com/sequana/sequana/issues/480](https://github.com/sequana/sequana/issues/480)
  - Set singularity hub (v2.4)

2.16.28 Prior 0.5.X

0.5.2

- **BUGS:**
  - cutadapt rule: remove the ‘--progress bar’ for now because of a bug in atropos (reported) that fails in the progress bar code

- **Updates:**
  - pipeline pacbio_qc: finalise output tree structure.
  - pipeline quality_control: add sanity check (thread must be >1 for atropos) and run fastqc on unmapped data (rather than mapped).
  - pin atropos version to 1.1.10 and added to requirements.txt
  - Fix parsing of atropos report
– Update FastQC significantly to use atropos FastqReader instead of pysam.FastxFile
– documentation for the installation (remove docker, add singularity)
– rule/module atropos: implement ability to parse json report from atropos https://github.com/sequana/sequana/issues/448
– rule fastqc: the log is now a variable. all pipelines using this rule have been updated to save the log in {sample}/logs/ instead of ./logs
– add polyT in TruSeq adapters

**News:**

– add Singularity container
– BAM class (bamtools module): add plotting methods (coverage, letters, indels)
– Add Cigar class (cigar module).
– Sequana: add option to switch on/off the tooltips
– rule cutadapt: (1) check whether thread is set to > 1. if not set to 2 (2) add --report-format to save reports in JSON and TXT

### 0.5.1

**BUGS:**

– Set -t thread options correctly in the different rules (e.g. cutadapt)
– pipeline variant_calling: fix the VCF inputs when snpeff is off . See https://github.com/sequana/sequana/issues/471
– pipeline quality_control. Fix regression bug introduced by the use of sambamba in the bwa_mem_dynamic rule (see https://github.com/sequana/sequana/issues/472)
– Fix wrong total bases values in summary report of the quality_control pipeline computed in FastQC class (see https://github.com/sequana/sequana/issues/470)
– pipeline pacbio_qc: hard-coded the number of threads to 4 otherwise may fail on clusters. Does not change the pipeline or analysis itself
– sequana_coverage: fix chromosome option.
– Fix genbank_parser when the genbank contains several concatenated genbank entries. This fixes the coverage reports CSV file that had missing annotations.
– Fix regression bug introduced in rule bwa_mem_dynamic that messed up R1 and R2 order as compared to samtools by using sambamba. Fixed by using -N parameter.
– Fix the -p option to be before the input whenever pigz is used in a rules. Indeed -p may be ignored otherwise e.g. on clusters.

**Updates:**

– add pacbio option in the mapping code
– pacbio_qc: fix pattern to filter input BAM files
– Speed up fastq_count (https://github.com/sequana/sequana/issues/465)
– bamtools module: speed up initialisation. add is_sorted method.
– bedtools: limit number of points to 1,000,000 in plot_coverage and set ylimits manually to 6 mean coverage. add __eq__ function. See #464 issue
– Repeats can handle FastA properly (not limited to first sequence anymore)
– sequana_mapping: add thread in samtools call

0.5.0 August 2017

Tag a stable release

0.4.2 August 2017

• Updates:
  – pipeline: variant calling cleanup and finalised
  – pipeline: denovo updated (busco) and cleanup and finalised
  – pipeline: pachio_qc finalised
  – pipeline: rnaseq: finalised
  – module pachio: speed up initialisation; add a random_selection method; add a summary method;
• NEWS:
  – Sequanix: can now load cluster config
  – new rules: busco, busco_analysis, canu
  – new pipeline: pachio_denovo
  – multiqc modules integrated in sequana. See Developer guide for details.
  – module snakertools: new function get_pipeline_stats
  – new gallery example with statistics about the pipelines
• CHANGES:
  – remove random() function from FastQ (useless and will be put in new module simulation)

0.4.1 July 2017

• Update of Variant calling and denovo pipelines with HTML report creation
• Fix #421 (check for dot command in sequanix)
• Fix #420 (sequanix browser on Mac)
• sequana_coverage #417 division by 0 fixed
• snpeff bugs for special genbank cases fixed
0.4 July 2017

- Master release for sequanix

0.3 April-June 2017

- **BUG FIXES:**
  - sequanix:
    - rulegraph issue on SLURM system. Avoid the os.chdir
    - fastq_samples/fastq module: fix histogram_gc_content maximum range
    - rulegraph rule: fix issue #405 (spaces in path to snakefile)
    - genome coverage was buggy for multi chromosome and circular option on. Fixed
    - adapters/expdesign modules: fixe the case of design files with same sample name and same index but different lanes.
    - sequana_coverage. Fix Issue #416 (float division by zero)
  
- **CHANGES:**
  - sequanix:
    - * snakemake output is now cleared when pressing RUN
    - quality_control pipeline: default to atropos instead of cutadapt for adapter trimming. Kraken: remove classified reads and keep unclassified. Unclassified reads are now compressed. unclassified reads that are also compressed now.
  
- **NEW:**
  - pacbio module: cleanup and add funcion to convert input BAM into Fasta
  - sequence module: Repeats class added
  - new Snakemake pipeline called qc_pacbio to perform quick QC and taxonomy analysis for pacbio
  - add ORD, CDS, GC SKEW in sequence module.

0.2. - March - April 2017

- **NEWS:**
  - RNA-seq pipeline added (single-end only, paired-end upcoming) including all indexes for RNA-seq
  - Hierarchical kraken available
  - add new standalone called sequana_fox to expose the pyqt5 browser.
  - Sequanix first release
  - final version of the variant calling, denovo, quality_control and rna-seq pipelines.

- **CHANGES:**
  - Sequanix/Sequana: - config file can have the yml extension (in addition to yaml) - dropdown widgets in the form based on the docstrings in the config file - can import config to override default sequana config file - subprocesses killed when the main pipeline is stopped
0.1.21 - Feb 2017

• NEWS:
  – add sequana_debug_level function at top level to switch verbosity of informative messages (default is WARNING).
  – add pacbio module #351
  – quality control pipeline: atropos can be used in place of cutadapt #346

• CHANGES:
  – Running Median is 10 times faster #345
  – sequana_coverage: (1) --file1 alone was not working (2) automatically copy cluster-config in working directory and update runme.sh accordingly #342
  – sequana standalone:
    * handles cluster_config Snakemake option
    * add error message when adapter name is incorrect
  – sequanix: the help dialog is now created inside designer and has a proper scrollable browser dialog. cluster_config Snakemake option is also handle.
  – Remove galleria JS lib and related files (htmltools)
  – sequana_coverage: add --logging-level option

• BUG:
  – Fix #352: allow gc window size to be even (warning is shown and +1 to window size)
  – Fix # 354: cutadapt report that was mixing up R1/R2 trimming in the images.
  – --output-directory in sequana_coverage was failing
  – in coverage, centralness was buggy (regression) and use number of ROIs instead of the total base length #347
  – Fix multi_report summary for single end case #349

0.1.20 - Feb 2017

• CHANGES:
  – remove pyquickhelper dependencies and add a simple rest2html function in misc module.

0.1.19 - Feb 2017

• CHANGES:
  – misc module: factorise on_cluster() function used in compressor scripts to be used in other tools such as sequanix
  – compressor: limits max number of jobs to 20 (can be bypass manually), prevent run on TARS if snakemake-cluster not provided.
  – rules:
* dag: now the snakemake is called inside a temporary directory to avoid clash with the current snakemake process. This avoid error message. Fixes https://github.com/sequana/sequana/issues/331

- `__init__` was optimized as well as many modules to make use of the lazy import mechanism. The reporting package is not part of the exposed module. So:

  ```python
  from sequana import BAMReport
  ```

  is now:

  ```python
  from sequana.reporting.report_bam import BAMReport
  ```

**NEWS:**

- Sequanix stable version
- add TrueSeq adaptors
- add lazy import mechanism to speed up the time to import sequana, which speeds up the `--help` in the standalone

---

**0.1.17/0.1.18 - Jan 2017**

**Main NEWS** The GUI was completed and the current pipelines stabilised (RNA-seq, quality control, variant calling). The test suite was switched from nosetests to pytest, in particular to perform tests more easily on the Qt GUI.

**BUG Fixes:**

- experimental design and adapters API simplified fixing a few bugs in the process. Doc and tested finalised.
- Fix cutadapt rules, which was not filling the fwd and rev properly anymore when using the design file.
- in sequana main script, `--reference` was used by quality_pipeline only. Now, available for all.
- Fix the main script for the reference in variant calling pipeline.

**CHANGES:**

- `sequana_compressor`: for conversion from e.g. gz to bz2, use a pipe instead of double IO. Updated docs and tests ready for production.
- sequana standalone: `--pattern` changed to `--input-pattern` `--output-directory` changed to `--working-directory`
- remove pipetools module (obsolet)
- GUI revisited with qt designer + can now also read any snakefile/config file combo (not just sequana pipelines)
- RULES: adapters can now use `adapter_type` without a design (fwd and rev gets filled automatically)

**NEWS:**

- add rubicon adapters
- add ability to read JSON in SequanaConfig
2.16.29 2016

0.1.16

• BUG Fixes:
  – Fix sequana_taxonomy (https://github.com/sequana/sequana/issues/308)
  – Fix typo in sequana_coverage for multiple chromosome (https://github.com/sequana/sequana/issues/307)

• NEWS:
  – SequanaConfig can read back a SequanaConfig instance
  – Added a DummyManager for minimalist manager to create reports

0.1.15

• CHANGES:
  – coverage: https://github.com/sequana/sequana/issues/302 add histogram, better stats table. add --output-directory
  – Update docker (add bowtie, subread, firefox)
  – snakertools:
    * empty strings are kept as empty strings (not None)
    * remove check() method in SequanaConfig
    * cleanup (removing of templates) can be switch off

0.1.14

• CHANGES:
  – fastqc.histogram_sequence_lengths (log2 scale to log10)
  – multi_summary fixed and available for the quality_control pipeline
  – sequana_compressor: add --keep-going option by default so that if a file fails, other independent files are processed.
  – snakertools:
    * remove SnakeMakeProfile (not used)
    * remove sequana_check_config (not used)
    * remove deprecated __get_tagname
    * remove ExpandedSnakefile since not required anymore
    * Fix sample_file2 option that was not encoded properly
    * PipelineManager and SequanaConfig use new yaml parser
  – sequana_coverage: -- add back the sample name as prefix of the HTML report name -- a BED with two coverage columns is now accepted -- --download-genbank option added
  – sequana_summary works for the quality_control pipeline
– Simplify combos of input_directory, input_patter, input_samples, the new possible mutually exclusive input parameters of sequana standalone and all pipelines.

• **BUGS:**
  – Kraken: if no reads classified at all, errors were raised and quality_control summary report would fail. This is fixed now with a "nodata" image being shown.

• **NEWS**
  – GUI (draft version)
  – fq.gz are now allowed in the pipelines and should be supported in the future
  – More tests in particular a ./test/pipelines/ new directory

**0.1.13**

• **CHANGES:**
  – revisited all pipelines so that they can work of multi samples.
  – quality_phix, quqlity and quality_taxon pipelines merged in quality_control pipeline
  – running meadian won't fail anymore with odd window size (we add +1)
  – rulegraph is used as well as dag to create figures of the pipelines

• **NEWS:**
  – compressor: includes dsrc format in addition to bz2 and gz
  – snakemake rule extension for sphinx
  – add a pipeline manager in snaketools to handle all pipelines
  – a designexp module to handle adapter design files

**0.1.12**

• **BUGS:**
  – Fix bug in cutadapt pipeline when there is no adapters. Force a dummy adapters (XXX) otherwise trimming is performed on read1 only

• **NEWS:**
  – compressor rule and script available.
  – coverage annotation
  – multiple_summary draft
0.1.11

- NEWS:
  - add a docker
  - sequana_summary standalone
  - sequana_mapping standalone
  - Module has an overview field
- BUG FIXES:
  - cutadapt report handles single-end tables. Fix the reverse complement adapter files for the paired-end case
- CHANGES:
  - sequana_standalone: final version with stats

0.1.10 - July 2016

- NEWS:
  - sequana_coverage standalone
  - de-novo pipeline
- CHANGES:
  - Remove AdapterDB, a draft version that uses Kraken to detect adapters. Not relevant anymore
  - config.yaml is now in each pipeline to have a simplified version
  - sequana can known use single_indexed or multiple_indexed adapters, which are also provided within sequana (Nextera and PCR free cases)
  - Release for production (quality_taxon pipeline)

0.1.7 to 0.1.9 - July 2016

- NEWS:
  - rule data added and used in phix_removal (fastq_sampling + raw data switch)
  - kmer module
  - sequana_taxonomy standalone
- CHANGES:
  - reports are now in ./sequana/reporting
  - MAJOR refactoring of report/ directories in all pipelines to make them independent from the temporary analysis, which can then be removed.
- BUGS:
  - Fix running median issue in bedtools (window size larger than contig size)
0.1.6 - June 2016

• NEWS:
  – KrakenDownload class: download kraken_toydv from sequana/data repository or minikraken into a local directory
  – New method in FastQC to show ACGT content
  – Genomecov renamed into GenomeCov
  – Update main script significantly to create multiruns and handle adapters
  – GC content and plot GC vs coverage added in GenomeCov

• CHANGES:
  – sequana_data by default looks into resources/testing directory
  – in fastq module: FastQC a bit faster and FastQRandom class removed
  – add a moving_average function in misc module

• BUGS:
  – sequana_data was showing __init__ and __pycache__ as possible data sets
  – databases: filelist as a list was not implemented
  – in fastq.FastQ extra_head in gzip mode was missing the last row

prior 0.1.5 June 2016

• NEWS
  – sequana_taxonomy standalone available (kraken + krona)
  – sequana standalone available
  – quality_taxon pipeline available
  – module coverage for theoretical computations
  – add gallery in the documentation

• CHANGES:
  – module vcf_to_snpeff renamed as snpeff

• BUG:
  – Fix bug in running median (shift)

2.17 Glossary

**BAI**  The index file for a file generated in the BAM format. (This is a non-standard file type.)

**BAM**  Binary version of the Sequence Alignment Map (SAM) format.

**BED**  Format that defines the data lines displayed in an annotation track.

**DSRC**  A compression tool dedicated to FastQ files

**FASTA**  FASTA-formatted sequence files contains either nucleic acid sequence (such as DNA) or protein sequence information. FASTA files store multiple sequences in a single file.
**Sequana, Release 0.14.0**

**GFF** General Feature Format, used for describing genes and other features associated with DNA, RNA and Protein sequences.

**JSON** A human-readable data serialization language commonly used in configuration files. See https://en.wikipedia.org/wiki/JSON

**Module** A directory that contains a snakemake rule and an associated README file. This is especially relevant for the Sequana pipelines. See *Developer guide*.

**SAM** Sequence Alignment Map is a generic nucleotide alignment format that describes the alignment of query sequences or sequencing reads to a reference sequence or assembly

**Snakefile** A file that contains one or several Snakemake rules

**VCF** Variant Call Format, for use with the variant calling pipeline

**YAML** A human-readable data serialization language commonly used in configuration files. See https://en.wikipedia.org/wiki/YAML

### 2.18 Indices and tables

- genindex
- modindex
- search
sequana.assembly, 80

sequana.bamtools, 81
sequana.bed, 151
sequana.bedtools, 88

sequana.canu_scanner, 160
sequana.cigar, 96
sequana.cnv, 160
sequana.compare, 144
sequana.contigs, 80
sequana.coverage, 98
sequana.criteria, 187

sequana.databases, 102
sequana.datatools, 162
sequana.demultiplex, 161

sequana.enrichment, 105

sequana.fasta, 110
sequana.fastq, 106
sequana.featurecounts, 112
sequana.freebayes_vcf_filter, 154

sequana.gff3, 151

sequana.iem, 105
sequana.itol, 159

sequana.kmer, 119
sequana.kraken.analysis, 119
sequana.kraken.multikraken, 124

sequana.mh, 192
sequana.misc, 149
sequana.mixture, 189
sequana.modules_report.bamqc, 157
sequana.modules_report.base_module, 156
sequana.modules_report.coverage, 158

sequana.pacbio, 127
sequana.phred, 136

sequana.resources.data, 163
sequana.ribodesigner, 139
sequana.rnadiff, 140
sequana.running_median, 145

sequana.sequence, 115
sequana.sniffer, 163
sequana.snpeff, 148
sequana.stats, 187

sequana.taxonomy, 124
sequana.tools, 150
sequana.trf, 161

sequana.utils.datatables_js, 163

sequana.viz.anova, 167
sequana.viz.corrplot, 168
sequana.viz.heatmap, 170
sequana.viz.hinton, 174
sequana.viz.hist2d, 175
sequana.viz.imshow, 177
sequana.viz.pca, 179
**INDEX**

<table>
<thead>
<tr>
<th>A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>(Coverage property), 99</td>
</tr>
<tr>
<td>accession_to_info()</td>
<td>(EUtilsTools method), 102</td>
</tr>
<tr>
<td>adapters</td>
<td>(in module sequana.resources.data), 163</td>
</tr>
<tr>
<td>add_code_section()</td>
<td>(SequanaBaseModule method), 156</td>
</tr>
<tr>
<td>add_command()</td>
<td>(ChromosomeCoverageModule method), 158</td>
</tr>
<tr>
<td>add_file()</td>
<td>(ITOL method), 160</td>
</tr>
<tr>
<td>add_flag_section()</td>
<td>(BAMQCModule method), 157</td>
</tr>
<tr>
<td>add_float_right()</td>
<td>(SequanaBaseModule method), 156</td>
</tr>
<tr>
<td>add_fotorama()</td>
<td>(SequanaBaseModule method), 156</td>
</tr>
<tr>
<td>add_gi_to_header()</td>
<td>(ENADownload method), 102</td>
</tr>
<tr>
<td>add_images_section()</td>
<td>(BAMQCModule method), 158</td>
</tr>
<tr>
<td>add_locus_in_fasta()</td>
<td>(SnpEff method), 148</td>
</tr>
<tr>
<td>AIC()</td>
<td>(in module sequana.criteria), 187</td>
</tr>
<tr>
<td>AICc()</td>
<td>(in module sequana.criteria), 188</td>
</tr>
<tr>
<td>Alignment</td>
<td>(class in sequana.bamtools), 81</td>
</tr>
<tr>
<td>alpha</td>
<td>(RNADiffResults property), 142</td>
</tr>
<tr>
<td>alpha</td>
<td>(RNADiffTable property), 143</td>
</tr>
<tr>
<td>annotate()</td>
<td>(Volcano method), 186</td>
</tr>
<tr>
<td>ANOVA</td>
<td>(class in sequana.viz.anova), 167</td>
</tr>
<tr>
<td>anova()</td>
<td>(ANOVA method), 167</td>
</tr>
<tr>
<td>anscombe()</td>
<td>(VST static method), 193</td>
</tr>
<tr>
<td>append_existing_database()</td>
<td>(Taxonomy method), 125</td>
</tr>
<tr>
<td>as_dict()</td>
<td>(Alignment method), 82</td>
</tr>
<tr>
<td>as_dict()</td>
<td>(Cigar method), 97</td>
</tr>
<tr>
<td>as_sequence()</td>
<td>(Cigar method), 97</td>
</tr>
<tr>
<td>as_tuple()</td>
<td>(Cigar method), 97</td>
</tr>
<tr>
<td>AT_skew</td>
<td>(DNA property), 116</td>
</tr>
<tr>
<td>attributes</td>
<td>(GFF3 property), 152</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BAI</td>
<td>219</td>
</tr>
<tr>
<td>BAM</td>
<td>219</td>
</tr>
<tr>
<td>BAM</td>
<td>(class in sequana.bamtools), 82</td>
</tr>
<tr>
<td>bam_analysis_to_json()</td>
<td>(SAMBAtools method), 83</td>
</tr>
<tr>
<td>bam_to_mapped_unmapped_fastq()</td>
<td>(in module sequana.tools), 150</td>
</tr>
<tr>
<td>BAMQCModule</td>
<td>(class in sequana.modules_report.bamqc), 157</td>
</tr>
<tr>
<td>BAMSimul</td>
<td>(class in sequana.pacbio), 127</td>
</tr>
<tr>
<td>bar_plot_contigs_length()</td>
<td>(Contigs method), 80</td>
</tr>
<tr>
<td>Barcoding</td>
<td>(class in sequana.pacbio), 129</td>
</tr>
<tr>
<td>barplot()</td>
<td>(StatsFile method), 161</td>
</tr>
<tr>
<td>barplot_count_ORF_CDS_by_frame()</td>
<td>(DNA method), 116</td>
</tr>
<tr>
<td>barplot_per_sample()</td>
<td>(StatsFile method), 161</td>
</tr>
<tr>
<td>barplot_summary()</td>
<td>(StatsFile method), 161</td>
</tr>
<tr>
<td>basic_stats()</td>
<td>(ChromosomeCoverageModule method), 158</td>
</tr>
<tr>
<td>BED</td>
<td>219</td>
</tr>
<tr>
<td>bed</td>
<td>(ChromosomeCov property), 90</td>
</tr>
<tr>
<td>BED</td>
<td>(class in sequana.bed), 151</td>
</tr>
<tr>
<td>begin_end_repeat_position()</td>
<td>(Repeats property), 117</td>
</tr>
<tr>
<td>BIC()</td>
<td>(in module sequana.criteria), 188</td>
</tr>
<tr>
<td>BOC</td>
<td>(ChromosomeCov property), 90</td>
</tr>
<tr>
<td>boxplot_classified_vs_read_length()</td>
<td>(Kraken-Results method), 122</td>
</tr>
<tr>
<td>boxplot_mapq_concordance()</td>
<td>(PacbioMappedBAM method), 130</td>
</tr>
<tr>
<td>boxplot_qualities()</td>
<td>(SAMBAtools method), 83</td>
</tr>
<tr>
<td>boxplot_quality()</td>
<td>(FastQC method), 109</td>
</tr>
<tr>
<td>boxplot_read_length_vs_passes()</td>
<td>(PacbioSubreads method), 133</td>
</tr>
<tr>
<td>build_kmer()</td>
<td>(in module sequana.kmer), 119</td>
</tr>
<tr>
<td>BUSCO</td>
<td>(class in sequana.assembly), 80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C3</td>
<td>(ChromosomeCov property), 90</td>
</tr>
<tr>
<td>C4</td>
<td>(ChromosomeCov property), 90</td>
</tr>
<tr>
<td>CanuScanner</td>
<td>(class in sequana.canu_scanner), 160</td>
</tr>
<tr>
<td>check()</td>
<td>(MetropolisHasting method), 192</td>
</tr>
<tr>
<td>check()</td>
<td>(Sequence method), 118</td>
</tr>
<tr>
<td>check_and_save_input_tables()</td>
<td>(RNADiffAnalysis method), 141</td>
</tr>
<tr>
<td>check_comparisons()</td>
<td>(RNADiffAnalysis method), 141</td>
</tr>
<tr>
<td>check_condition()</td>
<td>(RNADiffAnalysis method), 141</td>
</tr>
<tr>
<td>ChromosomeCov</td>
<td>(class in sequana.bedtools), 88</td>
</tr>
<tr>
<td>ChromosomeCoverageModule</td>
<td>(class in sequana.modules_report.coverage), 158</td>
</tr>
</tbody>
</table>
Cigar (class in sequana.cigar), 96
cigarstring (Cigar attribute), 97
circular (GenomeCov property), 95
cluster_probes() (RiboDesigner method), 140
clustering_needed() (RiboDesigner method), 140
Clustermap (class in sequana.viz.heatmap), 170
CNVnator (class in sequana.cnv), 160
column_method (Heatmap property), 174
column_metric (Heatmap property), 174
columns (Filtered_freebayes property), 154
comments (FastA property), 110
comparisons (RNADesign property), 140
complement() (Sequence method), 118
compress() (Cigar method), 97
compute_gc_content() (GenomeCov property), 95
compute_zscore() (ChromosomeCov method), 90
conditions (RNADesign property), 140
Contigs (class in sequana.contigs), 80
ContigsBase (class in sequana.contigs), 81
ContigsSpades (class in sequana.contigs), 81
copy() (DoubleThresholds method), 94
copy_file() (SequanaBaseModule method), 156
Corrplot (class in sequana.viz.corrplot), 168
count_lines() (FastQ method), 106
count_reads() (FastQ method), 106
Coverage (class in sequana.coverage), 98
coverage_barplot() (ChromosomeCoverageModule method), 158
coverage_plot() (ChromosomeCoverageModule method), 158
CoverageModule (class in sequana.modules_report.coverage), 159
CRAM (class in sequana.modules.bamtools), 82
create_chromosome_reports() (CoverageModule method), 159
create_chromosome_table() (CoverageModule method), 159
create_combobox() (SequanaBaseModule method), 156
create_datatable() (DataTable method), 164
create_embedded_png() (SequanaBaseModule method), 156
create_files_for_rnadiff() (GFF3 method), 152
create_hide_section() (SequanaBaseModule method), 156
create_html() (SequanaBaseModule method), 157
create_javascript_function() (DataTable method), 164
create_javascript_function() (DataTableFunction method), 165
create_link() (SequanaBaseModule method), 157
create_report_content() (BAMQCModule method), 158
create_report_content() (ChromosomeCoverageModule method), 158
create_report_content() (CoverageModule method), 159
create_taxonomy_file() (NCBITaxonomy method), 124
CS (class in sequana.bamtools), 82
CV (ChromosomeCov property), 90

D
DataTable (class in sequana.utils.datatables_js), 163
datatable_columns (DataTableFunction property), 165
datatable_options (DataTableFunction property), 165
DataTableFunction (class in sequana.utils.datatables_js), 164
df (BAMSimul property), 127
df (ChromosomeCov property), 90
df (Corrplot attribute), 169
df (DataTable property), 164
df (FeatureCount property), 112
df (Filtered_freebayes property), 154
df (GFF3 property), 152
df (Heatmap property), 174
df (IEM property), 105
df (KrakenResults property), 122
df (PacbioSubreads property), 133
df_shustring (Repeats property), 117
diagnostics() (MetropolisHasting method), 192
DNA (class in sequana.sequence), 115
do_merge (Repeats property), 117
DOC (ChromosomeCov property), 90
DoubleThresholds (class in sequana.bedtools), 93
download() (KrakenDownload method), 120
download_assembly_report() (NCBIDownload method), 102
download_fasta() (ENADownload method), 102
download_fasta_and_genbank() (in module sequana.snpeff), 149
download_genomes_from_ncbi() (NCBIUpload method), 102
download_ncbi_refseq_release() (NCBIUpload method), 102
download_taxdump() (NCBITaxonomyReader method), 104
download_taxonomic_file() (Taxonomy method), 125

DSRC, 219

E
EM (class in sequana.mixture), 189
ena_id_to_gi_number() (ENADownload method), 102
ENADownload (class in sequana.bamtools), 102
entropy() (DNA method), 116
estimate() (EM method), 189
estimate() (GaussianMixtureFitting method), 190
EUtilsTools (class in sequana.databases), 102
evenness (ChromosomeCov property), 90
evenness() (in module sequana.stats), 187
explode() (FastA method), 110
export() (ITOL method), 160
export_to_csv_bed() (RiboDesigner method), 140
export_tofasta() (RiboDesigner method), 140
extract_head() (FastQ method), 106

F

FASTA, 219
FastA (class in sequana.fasta), 110
FastQ (class in sequana.fastq), 106
FastQC (class in sequana.fastq), 108
feature_dict (GenomeCov property), 95
FeatureCount (class in sequana.featurecounts), 112
FeatureCountMerger (class in sequana.featurecounts), 112
features (GFF3 property), 152
fetch_by_id() (Taxonomy method), 125
fetch_by_name() (Taxonomy method), 126
fetch_clip() (in module sequana.cigar), 98
fetch_deletion() (in module sequana.cigar), 98
fetch_exon() (in module sequana.cigar), 98
fetch_intron() (in module sequana.cigar), 98
filter() (FastA method), 111
filter() (FastQ method), 107
filter() (RNADiffTable method), 143
filter_bool() (BAMSimul method), 127
filter_length() (BAMSimul method), 127
filter_length() (PacbioSimul method), 133
filter_mapq() (PacbioMappedBAM method), 130
filter_names_dmp_file() (NCBITaxonReader method), 104
filter_nodes_dmp_file() (NCBITaxonReader method), 104
filter_vcf() (VCF_freebayes method), 155
Filtered_freebayes (class in sequana.freebayes_vcf_filter), 154
filters_params (VCF_freebayes property), 155
find_taxon() (Taxonomy method), 126
findpos() (in module sequana.misc), 149
Fitting (class in sequana.mixture), 190
format_contigs_denovo() (FastA method), 111
frame (Heatmap property), 174
ftp_url (NCBITaxonReader attribute), 104

G

G (Coverage property), 99
GaussianMixtureFitting (class in sequana.mixture), 190
GaussianMixtureModel (class in sequana.mixture), 190
GC_content() (FastA method), 110
gc_content() (Sequence method), 118
GC_content_sequence() (FastA method), 110
GC_skew (DNA property), 116
gc_vs_coverage() (ChromosomeCoverageModule method), 158
gc_window_size (GenomeCov property), 96
genbank_filename (GenomeCov property), 96
genomized_anscombe() (VST static method), 193
GenomeCov (class in sequana.bedtools), 94
gctg_content() (FastQC method), 109
get_all_probes() (RiboDesigner method), 140
get_args() (DoubleThresholds method), 94
get_attributes() (GFF3 method), 153
get_average_name_per_taxon() (NCBITaxonReader method), 104
get_CDS_exons() (BED method), 151
get_centralness() (ChromosomeCov method), 91
get_children() (Taxonomy method), 126
get_complement() (Sequence method), 118
get_contig_per_chromosome() (Contigs method), 80
get_coverage() (PacbioMappedBAM method), 130
get_data_reads() (StatsFile method), 161
gt() (Contigs method), 80
gt() (MultiKrakenResults method), 124
gt() (MultiKrakenResults2 method), 124
gt() (SAMBAmbase method), 83
gt() (SAMBAmbase method), 83
gt() (SAMBAmbase method), 83
gt() (SAMBAmbase method), 83
gt() (SAMBAmbase method), 83
gt() (SAMBAmbase method), 83
gt() (SAMBAmbase method), 83
gtc() (CONTGSBase method), 81
gtc_content() (SAMBAmbase method), 84
gtc_correlation() (ChromosomeCov method), 91
gtc_lists() (RNASeqResults method), 142
gtc_guess() (Fitting method), 190
gtc_kmer() (in module sequana.kmer), 119
gtc_length_count() (SAMBAmbase method), 84
gtc() (FastQ method), 107
gtc() (SAMBAmbase method), 84
gtc() (SAMBAmbase method), 84
gtc() (SAMBAmbase method), 84
gtc() (SAMBAmbase method), 84
gtc() (SAMBAmbase method), 84
get_mapq_as_df() (SAMBAmbase method), 84
get_mapq_as_df() (SAMBAmbase method), 84
get_max_gc_correlation() (ChromosomeCov method), 91
get_mean_contig_length() (Coverage method), 99
get_mean_nb_passes() (PacbioSubreads method), 133
get_mean_number_contigs() (Coverage method), 99
get_mean_reads_per_contig() (Coverage method), 100
get_meaning() (SAMFlags method), 88
get_most_probable_strand() (in module sequana.featurecounts), 114
get_most_probable_strand_consensus() (in module sequana.featurecounts), 114
get_names_for_given_rank() (Taxonomy method), 126
get_number_of_ccs() (PacbioSubreads method), 133
get_occurences() (Sequence method), 118
get_parent_name() (Taxonomy method), 126
get_parent_taxon() (Taxonomy method), 126
get_percent_genome_sequenced() (Coverage method), 100
get_ranks() (Taxonomy method), 126
get_read_names() (SAMBAMbase method), 84
get_record_for_given_rank() (Taxonomy method), 126
get_required_coverage() (Coverage method), 100
get_reverse() (Sequence method), 118
get_reverse_complement() (Sequence method), 118
get_rna_pos_from_gff() (RiboDesigner method), 140
get_rois() (ChromosomeCov method), 91
get_samflags_count() (SAMBAMbase method), 84
get_samtools_stats_as_df() (SAMBAMbase method), 84
get_scientific_name() (NCBITaxonReader method), 104
get_size() (ChromosomeCov method), 91
get_specific_commons() (RNADiffResults method), 142
get_stats() (ChromosomeCov method), 91
get_stats() (FastA method), 111
get_stats() (FastQC method), 109
get_stats() (GenomeCov method), 96
get_stats() (SAMBAMbase method), 85
get_stats_full() (SAMBAMbase method), 85
get_summary() (ChromosomeCov method), 91
get_summary() (Coverage method), 100
get_summary_string() (BUSCO method), 80
get_table() (Coverage method), 100
get_taxon_from_scientific_name() (NCBITaxonReader method), 104
get_taxonomy_db() (KrakenResults method), 122
get_transcript_ranges() (BED method), 151
get_variants() (VCF_freebayes method), 155
getfile() (CanuScanner method), 160
GFF, 220
GFF3 (class in sequana.gff3), 151
GZLineCounter (class in sequana.tools), 150
H
hdtr (DoubleThresholds property), 94
header (IEM property), 105
header (Repeats property), 117
Heatmap (class in sequana.viz.heatmap), 171
heatmap() (RNADiffResults method), 142
heatmap_vst_centered_data() (RNADiffResults method), 142
high (DoubleThresholds property), 94
high2 (DoubleThresholds property), 94
hinton() (in module sequana.viz.hinton), 174
hist() (GenomeCov method), 96
Hist2D (class in sequana.viz.hist2d), 175
hist.cnvs() (TRF method), 161
hist_concordance() (PacbioMappedBAM method), 130
hist_contig_length() (ContigsSpades method), 81
hist_coverage() (SAMBAMbase method), 85
hist_entropy() (TRF method), 161
hist_event_size() (CNVnator method), 160
hist_GC() (BAMSimul method), 127
hist_GC() (PacbioMappedBAM method), 130
hist_GC() (PacbioSubreads method), 133
hist_length_repeats() (Repeats method), 117
hist_length_repetition() (TRF method), 161
hist_median_ccs() (PacbioMappedBAM method), 130
hist_nb_passes() (PacbioSubreads method), 133
hist_ORF_CDS_linearscale() (DNA method), 116
hist_ORF_CDS_logscale() (DNA method), 116
hist_period_size() (TRF method), 162
hist_plot_contig_length() (Contigs method), 81
hist_polymerase_per_barcode() (Barcoding method), 129
hist_mean_polymerase_read_length() (Barcoding method), 129
hist_median_ccs() (PacbioMappedBAM method), 130
hist_nb_passes() (PacbioSubreads method), 133
hist_ORF_CDS_linearscale() (DNA method), 116
hist_ORF_CDS_logscale() (DNA method), 116
hist_period_size() (TRF method), 162
hist_plot_contig_length() (Contigs method), 81
hist_polymerase_per_barcode() (Barcoding method), 129
hist_quality_per_barcode() (Barcoding method), 129
hist_read_length() (BAMSimul method), 128
hist_read_length() (CanuScanner method), 160
hist_read_length() (PacbioMappedBAM method), 131
hist_read_length() (PacbioSubreads method), 134
hist_read_length2() (CanuScanner method), 160
hist_repet_by_sequence() (TRF method), 162
hist_snr() (PacbioSubreads method), 134
hist_soft_clipping() (SAMBAMbase method), 85
hist_trimming_read_length() (CanuScanner method), 160
histo_classified_vs_read_length() (KrakenResults method), 122
histogram_gc_content() (FastQC method), 109
histogram_sequence_lengths() (FastQC method), 109
html_id (DataTable property), 164
html_id (DataTableFunction property), 165
I
Identifier (class in sequana.fastq), 109
IEM (class in sequana.iem), 105
import_tables() (RNADiffResults method), 142
imshow (class in sequana.viz.imshow), 177
imshow_anova_pairs() (ANOVA method), 168
include_svg_image() (SequanaBaseModule method), 157
index_adapters (IEM property), 105
infer_strandness() (SAMBAMbase method), 85
init() (NCBITaxonReader method), 104
init_roi_datatable() (CoverageModule method), 159
instrument (IEM property), 105
inverse_anscombe() (VST static method), 193
is_fastq() (in module sequana.fastq), 110
is_joint (VCF_freebayes property), 155
is_paired (SAMBAMbase property), 85
is_sorted (SAMBAMbase property), 85
ITOL (class in sequana.itol), 159

J
joining() (FastQ method), 107
JSON, 220

K
k (Fitting property), 190
keep_conditions() (RNADesign method), 140
kraken_to_csv() (KrakenResults method), 122
kraken_to_json() (KrakenResults method), 122
kraken_to_krona() (KrakenResults method), 122
KrakenAnalysis (class in sequana.kraken.analysis), 119
KrakenDB (class in sequana.kraken.analysis), 120
KrakenDownload (class in sequana.kraken.analysis), 120
KrakenPipeline (class in sequana.kraken.analysis), 120
KrakenResults (class in sequana.kraken.analysis), 121
KrakenSequential (class in sequana.kraken.analysis), 123

L
L (Coverage property), 99
launch_snpeff() (SnpEff method), 149
ldtr (DoubleThresholds property), 94
length (Repeats property), 117
lengths (Fasta property), 111
list_len_repeats (Repeats property), 117
load_records() (Taxonomy method), 126
to2_fc (RNADiffResults property), 142
to2_fc (RNADiffTable property), 143
to2_likelihood() (GaussianMixtureModel method), 190
longest_substring (Repeats property), 117
low (DoubleThresholds property), 94
low2 (DoubleThresholds property), 94

M
mean_quality (Quality property), 137
method (GaussianMixtureFitting property), 190
MetropolisHasting (class in sequana.mh), 192
Module, 220
module sequana.assembly, 80
sequana.bamtools, 81
sequana.bed, 151
sequana.bedtools, 88
sequana.canu_scanner, 160
sequana.cigar, 96
sequana.cnv, 160
sequana.compare, 144
sequana.contigs, 80
sequana.coverage, 98
sequana.critera, 187
sequana.databases, 102
sequana.datatools, 162
sequana.demultiplex, 161
sequana.enrichment, 105
sequana.fasta, 110
sequana.fastq, 106
sequana.featurecounts, 112
sequana.freebayes_vcf_filter, 154
sequana.gff3, 151
sequana.iem, 105
sequana.itol, 159
sequana.kmer, 119
sequana.kraken.analysis, 119
sequana.kraken.multikraken, 124
sequana.mh, 192
sequana.misc, 149
sequana.mixture, 189
sequana.modules_report.bamqc, 157
sequana.modules_report.base_module, 156
sequana.modules_report.coverage, 158
sequana.pacbio, 127
sequana.phred, 136
sequana.resources.data, 163
sequana.ribodesigner, 139
moving_average() (ChromosomeCov method), 91
moving_average() (in module sequana.stats), 187
mRNA_inner_distance() (SAMBAMbase method), 86
MultiFeatureCount (class in sequana.featurecounts), 112
MultiKrakenResults (class in sequana.kraken.multikraken), 124
MultiKrakenResults2 (class in sequana.kraken.multikraken), 124

N
N (Coverage property), 99
n_lines (FastQ property), 107
n_reads (FastQ property), 107
name (IEM property), 105
names (FastA property), 111
names (Repeats property), 117
NCBIDownload (class in sequana.databases), 102
NCBITaxonomy (class in sequana.taxonomy), 124
NCBITaxonReader (class in sequana.databases), 103
next() (ChromosomeCov method), 92
next() (FastA method), 111
next() (FastQ method), 107
normalized_coverage() (ChromosomeCoverageModule method), 158
normpdf() (in module sequana.misc), 149

O
order() (Corrplot method), 169
ORF_pos (DNA property), 116

P
PacbioMappedBAM (class in sequana.pacbio), 130
PacbioSubreads (class in sequana.pacbio), 132
params (Corrplot attribute), 169
pattern (Cigar attribute), 98
PBSim (class in sequana.pacbio), 129
PCA (class in sequana.viz.pca), 179
pdf() (GaussianMixtureModel method), 190
pie_plot() (BUSCO method), 80
plot() (Clustermap method), 171
plot() (CNVnator method), 160
plot() (Corrplot method), 169
plot() (EM method), 189
plot() (Fitting method), 190
plot() (Heatmap method), 174
plot() (Hist2D method), 177
plot() (Imshow method), 178
plot() (KrakenResults method), 122
plot() (PCA method), 179
plot() (Quality method), 139
plot() (Repeats method), 117
plot() (ScatterHist method), 180
plot() (Volcano method), 186
plot2() (KrakenResults method), 123
plot_acgt_content() (FastQC method), 109
plot_all_skews() (DNA method), 116
plot_and_save_all() (Barcoding method), 129
plot_bar_flags() (SAMBAMbase method), 86
plot_bar_mapq() (SAMBAMBase method), 86
plot_boxplot_normeddata() (RNADiffResults method), 142
plot_boxplot_rawdata() (RNADiffResults method), 142
plot_common_major_counts() (RNADiffCompare method), 144
plot_contig_length_vs_GC() (Contigs method), 81
plot_contig_length_vs_GC() (ContigsBase method), 81
plot_contig_length_vs_nreads() (Contigs method), 81
plot_correction_check1() (CanuScanner method), 160
plot_corrpplot_counts_normed() (RNADiffCompare method), 144
plot_corrpplot_counts_raw() (RNADiffCompare method), 144
plot_count_per_sample() (RNADiffResults method), 142
plot_coverage() (ChromosomeCov method), 92
plot_coverage() (SAMBAMBase method), 86
plot_dendogram() (RNADiffResults method), 142
plot_density() (RNADiffResults method), 142
plot_dispersion() (RNADiffResults method), 142
plot_feature_most_present() (RNADiffResults method), 142
plot_foldchange() (RNADiffCompare method), 144
plot_GC_read_len() (SAMBA MarginBase method), 86
plot_GC_read_len() (SAMBA MarginSimul method), 128
plot_GC_read_len() (PacbioMappedBAM method), 131
plot_GC_read_len() (PacbioSubreads method), 135
plot_gc_vs_coverage() (ChromosomeCov method), 92
plot_hist_coverage() (ChromosomeCov method), 92
plot_hist_normalized_coverage() (ChromosomeCov method), 92
plot_hist_zscore() (SAMBA MarginBase method), 86
plot_insert_size() (SAMBA MarginBase method), 87
plot_isomap() (RNADiffResults method), 142
plot_jaccard_distance() (RNADiffCompare method), 144
plot_kmer() (Canu.Scanner method), 160
plot_mds() (RNADiffResults method), 142
plot_padj_hist() (RNADiffTable method), 144
plot_pca() (RNADiffResults method), 142
plot_pca_vs_max_features() (PCA method), 179
plot_percentage_null_read_counts() (RNADiffResults method), 143
plot_polymerase_per_barcode() (Barcoding method), 129
plot_pvalue_hist() (RNADiffTable method), 144
plot_read_length() (SAMBA MarginBase method), 87
plot_rois() (ChromosomeCov method), 93
plot_scatter_contig_length_nread_cov() (Contigs method), 81
plot_stacked_hist() (MultiKrakenResults method), 124
plot_stacked_hist() (MultiKrakenResults2 method), 124
plot_strandness() (MultiFeatureCount method), 114
plot_subreads_histogram() (Barcoding method), 129
plot_unknown_barcodes() (StatsFile method), 161
plot_upset() (RNADiffResults method), 143
plot_venn() (in module sequana.viz.venn), 182
plot_venn_all() (RNADiffCompare method), 144
plot_venn_down() (RNADiffCompare method), 144
plot_venn_up() (RNADiffCompare method), 144
plot_volcano() (RNADiffCompare method), 145
plot_volcano() (RNADiffTable method), 144
plot_volcano_differences() (RNADiffCompare method), 145
png_to_embedded_png() (SequanaBaseModule method), 157
proba_to_quality_sanger() (in module sequana.phred), 139

Q
Quality (class in sequana.phred), 137

S
SAM, 220
sequences

module, 161

set_gene_lists()

module, 167

set_links_to_column()

module, 168

Sequence

module, 170

sequana.viz.heatmap

module, 174

sequana.viz.hist2d

module, 175

sequana.viz.imshow

module, 177

sequana.viz.pca

module, 179

sequana.viz.scatter

module, 180

sequana.viz.venn

module, 182

sequana.viz.volcano

module, 185

sequana.vst

module, 192

sequana.data() (in module sequana.datatools), 162

SequanaBaseModule (class in sequana.modu le_report.base_module), 156

Sequence (class in sequana.sequence), 117

sequence (Sequence property), 119

sequences (FastA property), 111

set_gene_lists() (RNADiffTable method), 144

set_links_to_column() (DataTableFunction method), 166

set_overlap_filtering() (CanuScanner method), 160

set_read_correction() (CanuScanner method), 161

set_tooltips_to_column() (DataTableFunction method), 166

settings (IEM property), 105

show() (KrakenPipeline method), 121

simulate() (MetropolisHasting method), 192

Snakefile, 220

sniffer() (in module sequana.sniffer), 163

SnPEff (class in sequana.snpeff), 148

split_chunks() (FastQ method), 108

split_lines() (FastQ method), 108

stats (PacbioSubreads property), 136

stats() (Cigar method), 98

stats() (Contigs method), 81

stats() (FastQ method), 108

stats() (Sequence method), 119

StatsBAM2Mapped (class in sequana.tools), 150

StatsFile (class in sequana.demultiplex), 161

STD (ChromosomeCov property), 90

stride() (PacbioSubreads method), 136

subcoverage()

(ChromosomeCoverageModule method), 158

summary (SAMBAMbase property), 87

summary() (BUSCO method), 80

summary() (FastA method), 111

summary() (PacbioSubreads method), 136

summary() (RNADiffResults method), 143

summary() (RNADiffTable method), 144

T

target_distribution() (PBSim method), 130

Taxonomy (class in sequana.taxonomy), 125

taxons (KrakenResults property), 123

template (RNADiffAnalysis attribute), 141

textwrap() (in module sequana.misc), 149

threshold (DNA property), 116

threshold (Repeats property), 117

thresholds (ChromosomeCov attribute), 93

to_bed() (GFF3 method), 153

to_csv() (ChromosomeCov method), 93

to_csv() (Filtered_freebayes method), 154

to_csv() (GenomeCov method), 96

to_csv() (RNADiffResults method), 143

to_fasta() (BAMSimul method), 128

to_fasta() (FastA method), 112

to_fasta() (FastQ method), 108

to_fasta() (GFF3 method), 153

to_fasta() (IEM method), 105

to_fasta() (PacbioMappedBAM method), 131

to_fasta() (PacbioSubreads method), 136

to_fastq() (BAMSimul method), 129

to_fastq() (PacbioMappedBAM method), 132

to_fastq() (PacbioSubreads method), 136

to_fastq() (SAMBAMbase method), 87

to_gtf() (GFF3 method), 153

to_html() (StatsBAM2Mapped method), 150

to_igv_chrom_size() (FastA method), 112

to_json() (KrakenResults method), 123

to_json() (StatsBAM2Mapped method), 150

to_kmer_content() (FastQ method), 108

to_krona() (FastQ method), 108

to_summary_reads() (StatsFile method), 161

to_tsv() (FeatureCountMerger method), 112

to_vcf() (Filtered_freebayes method), 154

transcript_to_gene_mapping() (GFF3 method), 153

TRF (class in sequana.trf), 161

type_filter (DNA property), 116

type_window (DNA property), 116

types (Cigar attribute), 98
U
upload() *(ITOL method)*, 160

V
validate() *(IEM method)*, 106
Variant *(class in sequana.freebayes_vcf_filter)*, 155
variants *(Filtered_freebayes property)*, 154
VCF, 220
vcf *(Filtered_freebayes property)*, 154
VCF_freebayes *(class in sequana.freebayes_vcf_filter)*, 154
version *(IEM property)*, 106
version *(KrakenDB property)*, 120
Volcano *(class in sequana.viz.volcano)*, 185
VST *(class in sequana.vst)*, 192

W
wget() *(in module sequana.misc)*, 149
window *(DNA property)*, 116
window_size *(GenomeCov property)*, 96

X
Xtarget *(MetropolisHasting property)*, 192

Y
YAML, 220
Ytarget *(MetropolisHasting property)*, 192

Z
zscore_distribution() *(ChromosomeCoverageModule method)*, 159