## CONTENTS

1 Warning 3

2 What is Sequana? 5

3 User guide and reference 11
   3.1 Installation 11
   3.2 Overview 16
   3.3 Tutorial 21
   3.4 Pipelines 28
   3.5 Gallery 31
   3.6 Case Examples 41
   3.7 Applications (standalone) 44
   3.8 Sequanix Tutorial 49
   3.9 Developer guide 67
   3.10 Rules 79
   3.11 References 80
   3.12 FAQS 173
   3.13 Changelog 178
   3.14 Glossary 194
   3.15 Indices and tables 195

Python Module Index 197

Index 199
Current version: 0.8.0, Mar 24, 2020

**Python version**  Python 3.5, 3.6 although most modules are Python2.7 compatible.

**Source**  See http://github.com/sequana/sequana.

**Issues**  Please fill a report on github

**How to cite**  For Sequana in general including the pipelines, please use


For the **genome coverage** tool (sequana_coverage), please cite:


For **Sequanix** (GUI for Snakemake pipeline), please cite:

March 2020  Important notes for developers. Major refactoring for version 0.8 now pushed on master branch. Please use this new branch to create your own branches. If you still want to use the previous version, please checkout the branch master_20_03_2020 instead.
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)

Currently, the available pipelines cover 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, RNA-seq. 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.
Chapter 2. What is Sequana?
Chapter 2. What is Sequana?
Sequana, Release 0.8.0

MainWindow

File  Option  Help

A - Sequana pipelines  B - Generic pipelines

1 - Pipeline selection  2 - Input data (directory or files)  3 - Working directory

Browse  ....kelaer/Temp/test/analysis

Pipeline control

Until  Starting  Local or cluster run?  local

Snakemake output  !Python shell  Logger  Config parameters

options  -0 6 -trim-n
quality  30
rev

tool  cutadapt

bwa_mem_phix

do  yes
index_algorithm  is
options  -T 30
reference_file  Browse  phiX174.fa
threads  4

Run  Stop  Unlock  Open Report  Save Config  Show Dag

62%

./auto_examples/images/sphx_glr_plot_qc_pacbio_002.png
3.1 Installation

If you are a developer, you would want to install Sequana from source. There are lots of dependencies that require compilation and may be time consuming. We therefore recommend the Anaconda solution. Sequana is indeed available on bioconda. Note, however, that releases of Sequana are also available on Pypi so you could also use pip.

If you just want to test Sequana or Sequanix or one of the Sequana standalone, we also provide Singularity containers. This is a great solution for reproducibility as well. Containers are available on https://singularity-hub.org/collections/114/.

3.1.1 Latest recommended installation for developers

To install Sequana, you can run:

```bash
conda install -c anaconda qt
conda install -c anaconda pyqt=5.6.0
python setup.py install
```

3.1.2 Overview of installation methods

We support 3 types of installations:

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):

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

   and can be used as follows (for example):

   ```bash
   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:

   ```bash
   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) and releases are posted on Pypi:
These three methods are detailed hereafter.

### 3.1.3 From bioconda (Recommended)

If you have not installed Sequana, be aware that it relies on many dependencies that needs to be compiled (i.e., it is 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. Yet, using conda, this process is simplified.

**Install conda executable**

In practice, we do use Anaconda. We recommend to install conda executable via the manual installer (download). 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:

```
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):

```
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:

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

You should see:

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

```
conda create --name sequana_env python=3.5
```

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

```
source activate sequana_env
```

Installation

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

```
conda install sequana
```

This does not provide all dependencies needed by the different pipelines. So, you may need to install extra packages as listed in this requirement file that can be used with conda:

```
conda install --file https://raw.githubusercontent.com/sequana/sequana/master/requirements_pipelines.txt
```

Additional tools may need to be installed. Thos large packages are stored in another requirements to keep the main distribution lighter:

```
conda install --file https://raw.githubusercontent.com/sequana/sequana/master/requirements_pipelines_extra.txt
```

3.1.4 From Pypi website (released source code)

If you do not want to use conda, we provide releases on the Python Package Index website (pip tool):

```
pip install sequana
pip install PyQt5
```

**Warning:** we do not support this methods but it should work. The main issues being that you will need to install the dependencies yourself. See hereafter for some of the tool used by the pipelines
3.1.5 From GitHub Source code

Finally, if you are a developer and wish to use the latest code, you can install sequana from source:

```bash
git clone git@github.com:sequana/sequana.git
cd sequana
python setup.py install
```

This should install most of the required dependencies. However, you may need to install more packages depending on the pipeline used. See hereafter.

3.1.6 Singularity

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. This should be solved in the near future.

First, install singularity (http://singularity.lbl.gov/). You must use at least version 2.4. We tested this recipe with version 2.4.2 (Dec 2017):

```bash
VERSION=2.4.2
git https://github.com/singularityware/singularity/releases/download/$VERSION/
  singularity-$VERSION.tar.gz
tar xvf singularity-$VERSION.tar.gz
cd singularity-$VERSION
./configure --prefix=/usr/local
make
sudo make install
```

Second, 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 omit 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:

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

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

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

Then, inside the container, install or fix the problem and type exit to save the container.
Note: method tested with success on Fedora 23, ubuntu and Centos 6.

See also:
Notes for developers about Singularity especially to get specific versions.

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

### 3.1.7 Notes about dependencies

When installing **Sequana** with conda and from the source, it should install all the Python dependencies and you should be ready to go to use the Sequana Python library.

However, note that most of the pipelines rely on extra dependencies that are not necessarily Python-based. For instance **bwa** is in C, others may be in R or perl.

The list of requirements is available in the source code:

```plaintext
https://raw.githubusercontent.com/sequana/sequana/master/requirements_pipelines.txt
```

and conda may be used to install those dependencies automatically:

```bash
conda install --file https://raw.githubusercontent.com/sequana/sequana/master/requirements_pipelines.txt
```

Otherwise you need to proceed to the installation of those dependencies by yourself.

Note: **atropos** is an alternative to cutadapt with additional options but same type of functionalities and arguments. We use version 1.0.23 and above though.

Note: the **denovo_assembly** pipelines uses Quast tool, which we ported to python 3.5 and was pulled on Quast official github page. This is not yet in bioconda but one can get it from the quast github (sept 2016). This is to be installed manually by users (due to licensing restrictions)

Note: For **GATK** (variant caller), please go to https://software.broadinstitute.org/gatk/download/auth?package=GATK and download the file GenomeAnalysisTK-3.7.tar.bz2 ; then type:

```bash
gatk-register GenomeAnalysisTK-3.7.tar.bz2
```
3.1.8 Docker containers for Sequana

We do not provide Docker containers anymore. However, since sequana is posted on bioconda, one can get some Docker containers. For example version 0.4.1 is available as explained here below. For full list please checkout https://quay.io/repository/biocontainers/sequana

Example: sequana_coverage

To pull a Sequana container (here version 0.4.1), use this type of command:

docker pull quay.io/biocontainers/sequana:0.4.1--py35_0

Checkout the quay.io website. After pulling the image above, you can use it as follows:

docker run -v $PWD:/home/default -it quay.io/biocontainers/sequana:0.4.1--py35_0

Warning: once in the docker shell, go to /home/default. Here, this directory is linked to your real directory where you type “docker run...” so what you modify here is directly reflected in your directory!

Assuming you have a BED file JB409847 in your directory, otherwise uncomment the commented line here below:

cd /home/default
export MPLBACKEND="agg"
# wget https://tinyurl.com/y9j69t3k -O JB409847.bed
sequana_coverage --input JB409847.bed
exit

Back on your local directory, you should now see a ./report directory with the results of the analysis.

3.2 Overview

Contents

- Overview
  - Sequana library
    - Example 1: running median on coverage
    - Example2: read a fastq file
  - Sequana standalones
  - Sequana pipelines
    - Manually
    - Using sequana standalone
    - 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.

The Tutorial, Pipelines, Gallery and Case Examples sections provide many examples on their usage.

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.

### 3.2.1 Sequana 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 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

**Example 2: 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.

Many more functionalities are available. The reference guide should help you.
3.2.2 Sequana standalones

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.

3.2.3 Sequana pipelines

In Sequana, in addition to the library and standalone applications, we also provide a set of pipelines (see Pipelines section). The coverage tools described so far do not have a dedicated pipeline but is part of a more general pipeline called Variant Calling. Instead of describing in details that pipeline, let us explain the way pipelines can be created and run.

Manually

Pipelines are made of a Snakefile (a Makefile using Python) and an associated config file. Pipelines can be downloaded from the Sequana pipeline directory as well as the config file named `config.yaml`.

Copy the pipeline (ending in `.rules`) and the configuration file in a local directory. The config file is a generic template file and some fields must be changed. For instance the beginning of the file looks like:

```yaml
# list of your input file
samples:
  file1: "%(file1)s"
  file2: "%(file2)s"
```

For pipelines that takes FastQ files as inputs, the string `%(file1)s` must be replaced by a valid filename. If you do not have a second file, remove the next line (file2). Other similar fields must be filled if required by the pipeline.

Then, a pipeline must be executed using the executable `snakemake`. If you choose the variant_calling pipeline, the file is executed as follows:

```
snakemake -s variant_calling.rules
```

This will search for the `config.yaml` file locally. One good feature is that if you interrupt the pipeline (or if it fails), you can fix the problem and re-run the command above without executing the parts of the pipelines that were successfully run. If you want to start from scratch, add `--forceall` option:

```
snakemake -s variant_calling.rules --forceall
```

See also:

`Pipelines` section for more information.
Using sequana standalone

An easier way to initialise a pipeline, is to use sequana executable. For instance for the variant calling:

```bash
sequana --pipeline variant_calling
  --input-directory data/ \
  --input-readtag _[12].fastq \
  --extention fastq.gz \
  --reference reference.fasta \
  --working-dir analysis

cd analysis
snakemake -s variant_calling.rules --stats stats.txt
```

This will automatically download the pipeline, config file and update the latter as much as possible.

See also:

*Applications (standalone) section*

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 tutorial_sequanix.

3.2.4 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

3.3 Tutorial

The following example will show how to run the 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.
3.3.1 Quality pipeline

Sequana comes with standalone applications and pipelines in the form of Snakefile (snakemake).

The following example will show how to initialise and run the quality control pipeline 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.

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

```
sequana --pipeline quality_control --working-directory TEST --adapters PCRFree
```

This command downloads the required configuration file(s) in particular the config file and the pipeline itself. This example should work out of the box but you may want to look at the configuration file config.yaml. For instance, you may want to change the reference to the phix (by default we use phix174.fa, which is provided in Sequana) or change the adapter_removal section to your needs (cutadapt parameters, in particular the forward and reverse complement list of adapters; None by default).

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

```
cd TEST
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.

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

```
open Hm2_GTGAAA_L005/report_qc_Hm2_GTGAAA_L005/summary.html
```

See User guide and reference

3.3.2 Taxonomy

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 a Sequana pipeline (see pipeline_taxon and sequana.kraken) or this standalone application:

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

where <database_path> must be replaced with the proper path.

Open the local HTML file krona.html. An example is available in Krona example
3.3.3 Variant calling

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 section.

Initialise the pipeline

Call sequana standalone as follows:

```
sequana --pipeline variant_calling --input-directory . --working-directory TUTORIAL
```

Or use Sequanix.

Go to the project directory

```
cd TUTORIAL
```

Get the genbank reference

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

```
from bioservices import EUtils
eu = EUtils()
data = eu.EFetch(db="nuccore", 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:

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

New in v0.10

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

```
from sequana.snpeff import download_fasta_and_genbank
download_fasta_and_genbank("K01711", "measles")
```
Get a snpEff config file and update it

Then you need to initialise a config file for snpEff tool:

```python
from sequana import snpeff
v = snpeff.SnpEff("measles.gbk")
```

Update the snpeff config file

Edit the config file `config.yaml` and add the filename `measles.gbk` in the snpEff section:

```yaml
# snpEff parameter
snpeff:
  do: yes
  reference: "measles.gbk"
```

and bwa_ref section:

```yaml
# Bwa parameter for reference mapping
bwa_mem_ref:
  reference: "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 – note that for short genomes, you may need to decrease the window size.

**Warning:** the mark_duplicates may be changed in the close future to use another tool.

Run the pipeline

```bash
snakemake -s variant_calling.rules --stats stats.txt -p -j 4 --forceall
```

3.3.4 De novo

The denovo_assembly pipeline can be initialised in the same way:

```bash
sequana --pipeline denovo_assembly --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)
3.3.5 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 --pipeline rnaseq --input-directory . --working-directory EXAMPLE
   --adapter-fwd GATCGGAAGAGCACACGTCTGAACTCCAGTCA --adapter-rev GTGACTGGAGTTCAGACGTGTGCTCTTCCGATC
```

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.

An alternative is to use Sequanix: GUI for snakemake workflows.

Go to the project directory

```
cd EXAMPLE
```

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.
Edit the config file(s)

Edit the config file config.yaml and fill the genome section:

```yaml
genome:
  do: yes
  genome_directory: Saccer3
  name: Saccer3
#path to index name
  fasta_file: Saccer3/Saccer3.fa
  gff_file: Saccer3/Saccer3.gff
  rRNA_file:
    rRNA_feature: "rRNA"
```

**Warning:** Note that fastq_screen is off by default. Indeed, Sequana does not provide a fastq_screen database so far. Therefore, if you want to run fastq_screen, please see the manual (https://www.bioinformatics.babraham.ac.uk/projects/fastq_screen/) and add the config file in the tool section.

Finally, also edit the multi_config.yaml file and replace:

```yaml
custom_logo: "Institut_Pasteur.png"
```

with yours or as follows (empty, not an empty string like "")

```yaml
custom_logo: 
```

**Note:** there are other places with hard-coded path but the corresponding sections are not used by default. If you decide to use them (e.g. fastq_screen), you will need to edit the configuration file accordingly.

Run the pipeline

On local:

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

On SGE cluster:

```bash
snakemake -s rnaseq.rules --stats stats.txt -p -j 12 --nolock --cluster-config json
--cluster "qsub -l mem_total={cluster.ram} -pe thread {threads} -cwd -e logs -o logs -V -b y "
```

On slurm cluster

```bash
sbatch snakemake -s rnaseq.rules --stats stats.txt -p -j 12 --nolock --cluster-config json
--cluster "sbatch --mem={cluster.ram} --cpus-per-task={threads} "
```
### 3.3.6 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:

```
cd data
```

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

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

In **exec** mode, this is even simpler:

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

or with pre-filled parameters:

```
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:

\[
\text{singularity exec -B /projets/1/data/:/data sequana.simg sequana_coverage \(\cdots\)}
\]

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.

3.4 Pipelines

In Sequana parlance, a pipeline is an application based on Snakemake that consists of a Snakefile and a configuration file.

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

Pipelines can be initialised and run via a command line interface called sequana but we would recommend to use Sequanix: GUI for snakemake workflows instead.

The following sections are dedicated to each pipeline.

3.4.1 Coverage

3.4.2 denovo_assembly

3.4.3 Quality control

3.4.4 RNA-seq

3.4.5 Small RNA-seq

3.4.6 Variant Calling

3.4.7 Compressor

Overview compressor can be used to compress/uncompress FastQ files recursively. It was designed to uncompress gzipped files and to compress them back into a bzip2 format. It was then extended to dsrc and non-compressed files. Supported formats are gz and bz2 and dsrc (http://sun.aei.polsl.pl/dsrc/download.html).

Input Any number of FastQ files (compressed or not)

Output The input files (compressed or not)

requirements gzip and bzip and their parallel versions (pigz and pbzip2) as well as dsrc (DNA compression tool).
Usage

A standalone named `sequana_compressor` is provided. Please see:

```
sequana_compressor --help
```

to get detailed information about the arguments. The following example converts all file ending in fastq.gz into a new compression format (bz2). Note that “fastq” before the extension is required:

```
sequana_compressor --source fastq.gz --target fastq.bz2
```

If you want to add recursivity, add the `--recursive` argument. On a distributed system (e.g. slurm), you should use the `--snakemake-cluster`. For example on SLURM, add:

```
--snakemake-cluster "sbatch --qos normal"
```

`compressor` allows one to go from one format in (fastq, fastq.gz, fastq.bz2, fastq.dsrc) to any format in the same list. So this is a fully connected network as shown below:

![compressor codecs](_static/compressor_codecs.png)

Here is another example:

```
sequana_compressor --source fastq.gz --target fastq.bz2 --threads 8
--snakemake-cluster "sbatch --qos normal" --recursive --jobs 20
```

The number of jobs is set to 4 by default and limited to 20 to have a reasonable IO access. You can use more using the `--bypass` argument. If nodes have 8 CPUs, use threads=8, this means 20 nodes will be used.

**Warning:** During the conversion, a `.snakemake` is created in each processed directory. If you interrupt the process, `snakemake` locks the directory. If you get an error message about locked directories, relaunch your previous command with `--unlock` to unlock the directories and start again.
Requirements

Parallel version of gzip and bzip, as well as dsrs:

- pigz
- bzip2
- dsrc
- bunzip2

Config file

In principle you should not use any config file if you use the standalone. Note, however, the format of the underlying config file (for pbzip2 and pigz, the number of threads is automatically set to the number of available threads).

```yaml
compressor:
  source: fastq.gz
  target: fastq.bz2
  threads: 4
  recursive: True
  verbose: True
```

DAG

```
ng_to_fastq
```

Rules used by the pipeline

Depending on the value of the target and source, only one rule is included in the pipeline. For example if your source is `fastq.gz` and the target `fastq.bz2`, the `gz_to_fastq` rule is included. Its documentation is here below:

Others similar rules that convert from one compressed format to another compressed formats are:
3.4.8 Quality control Pacbio

3.4.9 pacbio denovo

3.4.10 FastQC

3.5 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.

3.5.1 BAM module example

Plot histogram of MAPQ values contained in a BAM file

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

Get a data set (BAM file) for testing

```
from sequana import BAM, sequana_data
datatest = sequana_data('test.bam', "testing")
```

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

```
b = BAM(datatest)
b.plot_bar_mapq()
```
3.5.2 Quality histogram a la fastQC

Get a data set example

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

Create a FastQC instance

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

plot the histogram

```python
qc.boxplot_quality()
```
3.5.3 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")
```

Create a sequana.pacbio.BAMPacbio instance

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

```
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>
ImportError: cannot import name 'BAMPacbio'
```
plot the histogram of read length

```
qc.hist_len()
```

```
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 'qc' is not defined
```

plot the histogram of the SNRs for each base

```
qc.hist_snr()
```

```
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 'qc' is not defined
```

**Total running time of the script:** (0 minutes 0.005 seconds)

**Download Python source code:** plot_qc_pacbio.py

**Download IPython notebook:** plot_qc_pacbio.ipynb

### 3.5.4 Coverage module example

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

Reading input BED file

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

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

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

Out:

```
Genome length: 19795
Sequencing depth (DOC): 931.31
Sequencing depth (median): 988.00
Breadth of coverage (BOC) (percent): 96.60
Genome coverage standard deviation: 237.15
Genome coverage coefficient variation: 0.25
```

Compute running median and zscore telling the algorithm that the chromosome is circular.

```
chrom.running_median(n=5001, circular=True)
chrom.compute_zscore()
print(chrom.get_centralness())
```
### 3.5.5 Kraken module example

In Sequana, we provide tools to quickly assess the taxonomic content of a 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
  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
  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

```
from sequana import KrakenResults, sequana_data
test_file = sequana_data("test_kraken.out", "testing")
k = KrakenResults(test_file)
df = k.plot(kind='pie')
print(df)
```

Out:

```
          others            1.534356
Measles virus strain Edmonston-Zagreb         5.737158
Measles virus strain MVi/California.USA/8.04  6.337558
Measles virus genotypes and isolates       6.871247
```

(continues on next page)
Unclassified 12.274850
Measles virus 67.244830
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:

```python
from sequana import KrakenResults, sequana_data
test_file = sequana_data("test_kraken.out", "testing")
k = KrakenResults(test_file)
k.to_js(onweb=False)  # The output fileame is krona.html by default
```

An example is available in Krona example

**Total running time of the script:** (0 minutes 0.230 seconds)

**Download Python source code:** plot_kraken.py

**Download IPython notebook:** plot_kraken.ipynb

### 3.5.6 Pipeline statistics

First, let us get the data

```python
from sequana.snakeutils 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 case of.

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 0.121 seconds)
Download Python source code: plot_pipeline_stats.py
Download IPython notebook: plot_pipeline_stats.ipynb

3.5.7 Running median example

Plot running median on a data set
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()
legend()
ylim([10, 50])

Total running time of the script: (0 minutes 0.038 seconds)
Download Python source code: plot_running_median.py
3.6 Case Examples

3.6.1 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 data set, 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 data set (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 scores across all bases](image)
Quality pipeline

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

```bash
sequana --pipeline quality_control --input-dir <DIR1> --working-directory trimming
```

Then go to the project and execute the pipeline:

```bash
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 (referred 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 --pipeline variant_calling --input-dir <DIR2> --project 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.gbk", "w") as fout:
    fout.write(data.decode())
from bioservices import ENA
ena = ENA()
```

(continues on next page)
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.gbk")

Edit the config.yaml to change those sections:

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

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

Run the analysis:

```
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 sorfes. 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.

3.7 Applications (standalone)
3.7.1 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
### 3.7.2 sequana

**Deprecated** will be removed and replaced by Sequanix

**Overview** Creates project(s) to run a Sequana pipeline(s)

The `sequana` executable can be used to create pipelines (and associated config file). For example:

```
sequana --pipeline quality --file1 R1.fastq.gz --file2 R2.fastq.gz --project TEST
```

will create a directory called TEST with a few files such as `quality.rules`, `config.yaml`, a `runme.sh` and a `README` file. Valid pipelines can be found using:

```
sequana --show-pipelines
```

There are many more options and documentation. Please use the `--help` option for more information.

### 3.7.3 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`).

### 3.7.4 sequana_summary

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

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

3.7.6 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, ...).

3.7.7 fastq related: fastq_count

Description count number of reads and lines

Example:

```
fastq_count --input test.fastq.gz
```

3.7.8 fastq related: fastq_head

Description Extract head of a fastq files (zipped or not)

Example:

```
fastq_head --input input.fastq.gz --nlines 10000 --output output.fastq.gz
```
3.7.9 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

Please see *Compressor* for details.

3.8 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.
3.8.1 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.

3.8.2 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.

3.8.3 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
```
Fig. 1: Snapshot of the Sequanix graphical user interface (GUI)
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

Fig. 3: In the browser, you can either select an existing directory, or create a new one by clickin on the Create Folder (red ellipse on top left corner). Note that depending on your system, the layout may be different
Sequana, Release 0.8.0

3.8. Sequanix Tutorial

[Diagram of Sequanix (Sequana GUI) interface]

1. A - Sequana pipelines
   - Generic pipeline

2. 1 - Pipeline selection
   - Input data (directory of files)

3. 3 - Working directory

4. quality_control

Pipeline control

Is it a local or cluster run? local

Please, Check the snakemake dialog (Ctrl+O) to set the number of nodes in the local or cluster tabs

Snakemake output

IPython shell

Logger

Config parameters

bwa_mem_phix

do

index_algorithm

options

reference_file

threads

fastq_sampling

do

Runnable

25%
Fig. 2: In the input data section, you have to click on the red browser button to select the directory where to find the FastQ (gzipped) files.
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.

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.
Fig. 4: A dialog showing the DAG (directed acyclic graph) with dependencies in the analysis pipeline

**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 with pre-filled values. For instance, to pre-fill the input directory, the working directory and the pipeline itself start **Sequanix** as follows:

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

For help, please type:

```bash
sequanix --help
```

**3.8.4 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
1 import glob
2 import os
3 (continues on next page)
```
# 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
```

3.8.5 **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

cfgfile: "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**
3.8. Sequanix Tutorial
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.
3.8.6 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. 5: 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 fire-
fox, 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 analaysis 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
Fig. 6: 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. 7: 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. 8: 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. 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.
Fig. 9: In the General tab, check boxes related to Snakemake are available. Any other options can be set in the editable line at the bottom.
**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**

### 3.8.7 FAQS

**How to run Sequanix on a SLURM cluster.**

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

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

Once connected, on a slurm system type:

```
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.

### 3.9 Developer guide

This section is a tutorial for developers who wish to include a Snakemake pipeline in Sequana.

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.

### 3.9.1 How to write a new module

A *Module* (in Sequana parlance) is a directory with a set of files: a Snakemake file (also known as *Snakefile*), a README file for the documentation and a configuration file (optional). The Snakefile may be a simple snakemake rule or a set of them (a pipeline).

#### 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 used using:

```python
>>> import sequana
>>> "count" in sequana.modules.keys()
False
```

So, let us name it `count`

#### Create a Snakefile rule

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).

---

### Store the rule in a Sequana module

We now store this Snakefile in the proper place. All modules are placed either in `./sequana/pipelines` or in `./sequana/rules` directory. The tree structure looks like:

```
--- rules
  | --- count
  | | --- count.rules
  | | --- README.rst
  --- pipelines
  | --- count_pipeline
  | | --- count_pipeline.rules
  | | --- README.rst
```

We have created a `count` directory in `./rules` and put the Snakefile in it (named `count.rules`).

A few comments:

- directory names must match the rule filename contain in it
- all Snakefiles end in `.rules`
- a `README.rst` must be present in all pipelines sub-directories

The README file in the rules can be empty. However, the README in the pipelines’s directory is used in the documentation and automatically parsed. See *The pipeline README file* section for further details.

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

```bash
>>> import sequana
>>> "count" in sequana.modules.keys()
True
```

---

### 3.9.2 Convention to design a rule

#### Use variables

Consider this Snakefile:

```python
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`. 

---

### 3.9. Developer guide
**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:

```
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 it 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

(continues on next page)
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)
```

3.9.3 How to write a new pipeline

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
configfile: "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"
```

3.9. Developer guide
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):

```plaintext
: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
~~~~~~~~~~~~~~
.. include:: ../sequana/pipelines/pipeline_count/requirements.txt

.. image:: https://raw.githubusercontent.com/sequana/sequana/master/sequana/pipelines/
    pipeline_count/dag.png

Details
~~~~~~~

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

count rule
^^^^^^^^^^

.. snakemakerule:: 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
#    item1: 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
#
# 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:

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

**3.9.4 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:

```
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:

```
python setup.py test
```

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

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

### 3.9.5 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 created sortable table in HTML using javascript behind the scene.

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

    js = table.create_javascript_function()
    html_tab = table.create_datatable(float_format='\$3gf')
    html = "{} {}"
    format(html_tab, js)
    self.sections.append({
        "name": "Table",
        "anchor": "table",
```

(continues on next page)
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
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()
```

Here is the full working example:
```python
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']
}

js = table.datatable.set_links_to_column('url', 'data')

html = "{} {}".format(html_tab, js)

self.sections.append({
    "name": "Table",
    "anchor": "table",
    "content": html
})


```

When using this module, one creates an HTML page called `mytest.html`. An instance of the page is available here: `report_example.html`
3.9.6 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
make html
```

3.9.7 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/](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`

3.9.8 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) !! Althoug 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.

### 3.10 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 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, . . . ).
3.11 References

Contents

- References
  - Adapters
  - Assembly related
  - CIGAR tools
  - BAMTOOLS related
  - BEDTOOLS related (coverage)
  - Coverage (theoretical)
  - Access to online database (e.g. ENA)
  - Experimental design
  - FASTQ module
  - FASTA module
  - Sequence module
  - Kmer module
  - ITOL module
  - IOTools module
  - Taxonomy related (Kraken - Krona)
  - Pacbio module
  - Phred quality
  - Running median
  - Snakemake module
  - Snpeff module
  - General tools
  - VCF module
  - Module Reports
  - Others
    * data related
    * report related
3.11.1 Adapters

Utilities to manipulate adapters

Adapters removal can be performed by many different tools such as CutAdapt, AlienTrimmer, Trimmomatic. Unfortunately, they tend to use different formats from FASTA to text files. Moreover, outputs are generally also reported in different formats.

Tools to extract specific adapters from FASTA files would also be handy. For instance, you may have all your adapters in a single file.

In this module, we provide:

- tools to manipulate adapters stored in Fasta format (`AdapterReader`).
- tools to export Fasta files with adapter content into other formats required by various adapter removal software
- A tool used to extract adapters from a FASTA file given their identifier, or sequence `FindAdaptersFromDesign`.

Our convention is to store list of adapters in FASTA format, which can be read using the `AdapterReader`:

```python
from sequana import sequana_data, AdapterReader
filename = sequana_data("adapters_Nextera_fwd.fa")
ar = AdapterReader(filename)
ar.get_adapter_by_index("N501")
```

Given a design file (see mod:`sequana.expdesign`), and a name for the type of adapters, one can easily extract the subset of relevant adapters to be used for a sample. Currently, the following set of adapters/design are available:

- Nextera single and double indexing
- Rubicon single indexing
- PCRFree single indexing
- TruSeq
- NEBNext single and double indexing

Note that TruSeq index 17, 24, and 26 are missing. This is normal. Those are “reserved” Illumina index.

For instance given a design file that gives the mapping between samples and a set of Nextera adapters, one would use:

```python
>>> from sequana import *
>>> filename = sequana_data("test_expdesign_hiseq.csv")
>>> design = ExpDesignAdapter(filename)
>>> fa = FindAdaptersFromDesign(design, "PCRFree")
>>> print(fa.sample_names[0])
'553-iH2-1'
>>> fa.get_adapters_from_sample("553-iH2-1")
```

See `FindAdaptersFromDesign` for details.

```python
class Adapter (identifier, sequence=None, comment=None)
    Class to store one adapter
    An adapter is just a sequence from a FASTA file. It contains an identifier, a sequence and possibly a comment.
```

**Warning:** The identifier provided must not contain the starting “>” character, which is added automatically when needed.

3.11. References
One can check if an adapter is equal to another. Only the sequence is checked for equality though.

Some Sequana notation have been added in the identifier to ease retrieval of index’s name and index’s sequence:

\texttt{\textbackslash \textgreater \texttt{NextFlex\_PCR\_Free\_adapter1|name:1|seq:CGATGT}}

Of course the string CGATGT must be found in the sequence itself.

\begin{verbatim}
ar = AdapterReader(sequana_data("adapters\_PCRFree\_fwd.fa"))
adapter = Adapter(ar[0])
adapter.identifier
adapter.comment
adapter.index_sequence
adapter.sequence
adapter.name
\end{verbatim}

\begin{verbatim}
@property comment
R/W adapter’s identifier

@property identifier
R/W adapter’s identifier

@property index_sequence
Read only access to the index sequence

@property name
Read only access to the index name

@property sequence
R/W adapter’s sequence
\end{verbatim}

class AdapterReader (filename)
Reader of FASTA file dedicated to adapters

A Fasta is just a set of this kind of paired-lines:

\begin{verbatim}
>\texttt{\textgreater \texttt{Nextera\_index\_N501|name:N501|seq:ACGT optional comment}}
ACGTACGTACGT
\end{verbatim}

where the \textit{optional comment} is separated from the identifier by a tabulation.

In the FASTA identifier, the first pipe delimits the official name (left hand side) from the name tag. The information on this example may be redundant but the \texttt{name} will be used throughout the \texttt{Sequana} code to ensure reproducibility.

\textbf{Note:} sequences are all in big caps.

\textbf{Note:} the universal adapter has no index so does not need to have the any tags for the name of index sequence. However, it must be called \texttt{Universal\_Adapter}

\begin{verbatim}
>>> from sequana import sequana_data, AdapterReader
>>> filename = sequana_data("adapters\_Nextera\_fwd.fa")
>>> ar = AdapterReader(filename)
>>> candidate = ar.get_adapter_by_index_name("S505")
>>> print(candidate[0])
>\texttt{\textgreater \texttt{Nextera\_index\_S505|name:S505|seq:GTAAGGAG}}
\end{verbatim}

(continues on next page)
AATGATACGGCGACCACCGAGATCTACACGTAAGGAGTCGTCGGCAGCGTC

>>> len(ar)
56

**Note:** Checks for uniqueness of the identifiers. If not unique, an error is raised

**Sources** document illumina #1000000002694 v01


**Constructor**

**Parameters** filename *(str)* – the input FASTA file

**property comments**

**get_adapter_by_identifier**(self, text)

Return adapter whose identifier matches the user text

**Parameters** index_identifier – the unique index identifier to be found. If several sequence do match, this is an error meaning the fasta file with all adapters is not correctly formatted.

**Returns** the adapter that match the index_name (if any) otherwise returns None

**get_adapter_by_index_name**(self, index_name)

Return adapter for the index name provided

Can be used only if the identifier contains the tag:

| name:an_index_to_be_found

For instance:

```
>Nextera_blabal|name:N505|seq:ACGT
>Nextera_blabal|seq:ACGT|name:N505
>Nextera_blabal|name:N505
```

are valid identifiers

**get_adapter_by_index_seq**(self, index_name)

See **get_adapter_by_index_name**(.)

**get_adapter_by_sequence**(self, subsequence)

Return one or several adapters with sub-sequence in their sequence

**Parameters** subsequence *(str)* – a string (ACGT letters)

**Returns** name and sequence in FASTA format that have the user sequence contained in their sequence

If the subsequence is short, it may return more than 1 adapters. Besides, the sequence is searched for without position information right now.

**property identifiers**
property index_names

property index_sequences

reverse(self)

Reverse all sequences inplace

```python
>>> from sequana import sequana_data, AdapterReader
>>> filename = sequana_data("adapters_Nextera_fwd.fa")
>>> filename2 = sequana_data("adapters_Nextera_rev.fa")
>>> ar = AdapterReader(filename)
>>> ar2 = AdapterReader(filename2)
>>> ar.reverse()
>>> ar == ar2
True
```

reverse_complement(self)

Reverse-complement all sequences inplace

```python
>>> from sequana import sequana_data, AdapterReader
>>> filename = sequana_data("adapters_Nextera_fwd.fa")
>>> filename = sequana_data("adapters_Nextera_revcomp.fa")
>>> ar = AdapterReader(filename)
>>> ar.reverse_complement()
>>> ar.to_fasta()
>>> ar == ar2
```

property sequences

to_dict(self)

Returns dictionary with key as identifier and values as list with comments and sequences

to_fasta(self, filename)

Save sequences into fasta file

class FindAdaptersFromDesign (design_filename, adapters)

Extract adapter(s) corresponding to an experimental design file

Used by sequana main script to build the adapter files for multi-samples projects as input to various adapter removal software.

Constructor

Parameters

- **design_filename** (*str*) – a CSV file that is compatible with our `sequana.expdesign.ExpDesignAdapter`

- **adapters** – the type of adapters (PCRFree, Nextera, Rubicon, TruSeq, SMARTer, Small)

The files of adapters are stored in Sequana and accessible with the `sequana_data` function. So, for instance if adapters is set to Nextera, the following file is used to identify the adapters:

```python
sequana_data("adapters_Nextera_fwd.fa")
```

New adapters files can be added on request. See resources/data/adapters for the full list. You can also use:

```python
from sequana.adapters import _get_registered_adapters
_get_registered_adapters()
```
check (self)

**get_adapters_from_sample** (self, sample_name)

Return a dictionary with adapters corresponding to the sample name

**Parameters**

- **sample_name** (str) – a valid sample name as found in the design file. One can check the content of the **sample_names** attribute.

**Returns**
a dictionary with the adapters in forward and reverse complement for index1 and index2 (if relevant).

```plaintext
>Universal_Adapter|name:universal
AATGATACGGCGACCACCGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCT
>Nextera_transposase_seq_1|name:transposase_seq_1
TCGTCGCGCCGTCAGATGTGTATAAGAGACAG
>Nextera_transposase_seq_2|name:transposase_seq_2
GTCTCGTGGGCTCGAGATGTGTATAAGAGACAG
>Nextera_index_N501|name:N501|seq:TAGATCGC
```

If sample name is given, it will figure out that for instance the index N501 is required. In addition, all sequences with identifier without the |seq: tag in their name will be added; So, here we will also have the universal and the two transposases.

**get_sample** (self, sample_name)

Return basic info about the sample name (from the design file)

**property sample_names**

return all sample names contained in the design file

**save_adapters_to_fasta** (self, sample_name, output_dir='. ')

Get index1, index2 and other standard adapters

**adapter_removal_parser** (filename)

Parses output of AdapterRemoval software

```python
>>> from sequana import adapters, sequana_data
>>> data = sequana_data("test_adapter_removal_output.txt", "testing")
>>> results = adapters.adapter_removal_parser(data)
>>> results['adapter1']
'AGATCGGAAGACACGTCTCAATTTGGCTATCTACAGTACACTGTCACTATAGGGAGACAG'
```

**fasta_fwd_rev_to_columns** (file1, file2=None, output_filename=None)

From 2 FASTA files (reverse and forward) adapters, returns 2-columns file

This is useful for some tools related to adapter removal that takes as input this kind of format

**Parameters**

- **filename1** (str) – FASTA format
- **filename2** (str) – FASTA format (optional)

The files must have a one-to-one mapping

**get_sequana_adapters** (type_, direction)

Return path to a list of adapters in FASTA format

**Parameters**

- **tag** – PCRFree, Rubicon, Nextera
- **type** – fwd, rev, revcomp

**Returns**
path to the adapter filename
### 3.11.2 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 retrieve 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 (self)**

**pie_plot (self, 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 (self, 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 (self)**

Return summary information of the missing, completed, fragmented orthologs

### 3.11.3 CIGAR tools

**class Cigar (cigarstring)**

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

```python
>>> 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 = ??? Last item is I + S + X !!! dans BWA, mismatch (X) not provided... should be deduced from last item - I - S

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 IS100Y is 1 since Y is not correct.

as_dict (self)
Return cigar types and their count

Returns dictionary

Note that repeated types are added:

```python
>>> c = Cigar('1S2M1S')
>>> c.as_dict()
{'S':2, 'M':2}
```

as_sequence (self)

as_tuple (self)
Decompose the cigar string into tuples keeping track of repeated types

Returns tuple

3.11. References
Sequana, Release 0.8.0

>>> from sequana import Cigar
>>> c = Cigar("1S2M1S")
(('S', 1), ('M', 2), ('S', 1))

cigarstring
the CIGAR string attribute

compress (self)
1S1S should become 2S. inplace modification

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

stats (self)
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

3.11.4 BAMTOOLS related

Tools to manipulate BAM/SAM files

<table>
<thead>
<tr>
<th>Alignment (alignment)</th>
<th>Helper class to retrieve info about Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAM (filename, *args)</td>
<td>BAM reader.</td>
</tr>
<tr>
<td>CRAM (filename, *args)</td>
<td>CRAM Reader.</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>
</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 BAM (filename, *args)
BAM reader. See SAMBAMBase for details

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
as_dict(self)
```

**class SAMFlags(value=4095)**

Utility to extract bits from a SAM flag

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

**Sequana, Release 0.8.0**

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

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

**class CS**(tag)
Interpret CS tag from SAM/BAM file tag

```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 CRAM**(filename, *args)
CRAM Reader. See SAMBAMBase for details

### 3.11.5 BEDTOOLS related (coverage)

Utilities for the genome coverage

**class GenomeCov**(input_filename, genbank_file=None, low_threshold=-4, high_threshold=4, idtr=0.5, hidtr=0.5, force=False, chunksize=5000000, quiet_progress=False, chromosome_list=[])
Create a list of dataframe to hold data from a BED file generated with samtools depth.

This class can be used to plot the coverage resulting from a mapping, which is stored in BED format. The BED file may contain several chromosomes. There are handled independently and accessible as a list of ChromosomeCov instances.

Example:

```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
gencov.thresholds.high = 4
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()
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 genomecov 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 analyze. If a chromosome is larger than the chunk size, it is split into N chunks. The segments are analyzed 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 analyzed 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** (self, 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 feature_dict
Get the features dictionary of the genbank.

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

property genbank_filename
Get or set the genbank filename to annotate ROI detected with ChromosomeCov.get_roi(). Changing the genbank filename will configure the GenomeCov.feature_dict.

get_stats(self)
Return basic statistics for each chromosome

Returns dictionary with chromosome names as keys and statistics as values.

See also:
ChromosomeCov.

Note: used in sequana_summary standalone

hist(self, logx=True, logy=True, fignum=1, N=25, lw=2, **kwargs)

input_filename = None
# check if the input is a csv of a previous analysis try:
    self.chr_list = None self._read_csv(input_filename) self.positions = {} #for chrom in self.chrom_names: #self.positions[chrom] = {“start”:start “end”:end “N”: N}
except FileNotFound as e: print(“FileNotFound error([0]): [1]”.format(e.errno, e.strerror)) sys.exit(1)

to_csv(self, output_filename, **kwargs)
Write all data in a csv.

Parameters

• output_filename (str) – csv output file name.
• kwargs (**) – parameters of pandas.DataFrame.to_csv().

property window_size
Get or set the window size to compute the running median. Size must be an interger.

class ChromosomeCov(genomcov, chrom_name, thresholds=None, chunksize=5000000)
Factory to manipulate coverage and extract region of interests.

Example:

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()
Chapter 3. User guide and reference
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}
\begin{itemize}
\item \texttt{df} – dataframe with position for a chromosome used within \texttt{GenomeCov}. Must contain
the following columns: [“pos”, “cov”]
\item \texttt{genomcov} –
\item \texttt{chrom\_name} – to save space, no need to store the chrom name in the dataframe.
\item \texttt{thresholds} – a data structure \texttt{DoubleThresholds} that holds the double threshold
values.
\item \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 \texttt{running\_median} and \texttt{compute\_zscore} functions.
\end{itemize}

\textbf{property BOC}
\hspace{1cm} breadth of coverage

\textbf{property C3}

\textbf{property C4}

\textbf{property CV}
\hspace{1cm} The coefficient of variation (CV) is defined as sigma / mu

\textbf{property DOC}
\hspace{1cm} depth of coverage

\textbf{property STD}
\hspace{1cm} standard deviation of depth of coverage

\textbf{property bed}

\textbf{compute\_zscore (self, k=2, use\_em=True, clip=4, verbose=True)}
\hspace{1cm} Compute zscore of coverage and normalized coverage.

\textbf{Parameters}
\begin{itemize}
\item \texttt{k (int)} – Number gaussian predicted in mixture (default = 2)
\item \texttt{clip (float)} – ignore values above the clip threshold
\end{itemize}

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

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

\textbf{property df}

\textbf{property evenness}
\hspace{1cm} Return Evenness of the coverage

\section*{3.11. References}

97
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 (self, 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 (self)
get_gc_correlation (self)
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)

get_max_gc_correlation (self, 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.

get_rois (self)
Keep positions with zscore outside of the thresholds range.

Returns a dataframe from FilteredGenomeCov

Note: depends on the thresholds low and high values.

get_size (self)
get_stats (self)
Return basic stats about the coverage data
only “cov” column is required.

Returns dictionary

get_summary (self, C3=None, C4=None, stats=None, caller='sequana.bedtools')
moving_average (self, 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

Store the results in the df attribute (dataframe) with a column named ma.
**plot_coverage** *(self, 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_label** – label for the running median
- **th_lw** – line width of the thresholds
- **th_color** – line color of the thresholds
- **th_ls** – line style 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,

In addition to the coverage, the running median and coverage confidence corresponding to the lower and upper zscore thresholds are shown.

**Note:** uses the thresholds attribute.

**plot_gc_vs_coverage** *(self, filename=None, bins=None, Nlevels=None, fontsize=20, norm='log', vmin=0, vmax=100, contour=True, cmap='BrBG', **kwargs)*

Plot histogram 2D of the GC content versus coverage

**plot_hist_coverage** *(self, 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** *(self, filename=None, binwidth=0.05, max_z=3)*

Barplot of the normalized coverage with gaussian fitting

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

Barplot of the zscore values

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

**reset** *(self)*

**property rois**

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

**running_median** *(self, 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.

```python
thresholds = None
try: self.thresholds = thresholds.copy()
except: self.thresholds = DoubleThresholds()
```

**to_csv** *(self, 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.

```python
dt = DoubleThresholds(-3,4,0.5,0.5)
```

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.

**copy** *(self)*
get_args(self)
property hdtr
property high
property high2
property ldtr
property low
property low2

3.11.6 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 nucleotides 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:

```python
>>> cover = Coverage(G=1000000, L=100)
>>> cover.N = 100000  # number of reads
10
>>> cover.a = 50  # What is the mean coverage
500000
```

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 $N$ reads 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:

\[ a = \log\left(-1/(E - 1)\right) \]

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** *(self)*

- Expected length of the contigs

\[
\frac{e^a - 1)L}{a}
\]

**get_mean_number_contigs** *(self)*

- Expected number of contigs

A binomial distribution with parameters \(N\) and \(p\)

\[(aG/L) \exp^{-a}\]

**get_mean_reads_per_contig** *(self)*

- Expected number of reads per contig

Number of reads divided by expected number of contigs:

\[N/(N \exp^{-a}) = e^a\]

**get_percent_genome_sequenced** *(self)*

- Return percent of the genome covered

\[100(1 - \exp -a)\]
get_required_coverage (self, 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

\( M (\text{float}) \) – 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

![Graph showing the relationship between uncovered genome and required coverage](image)

# The inverse equation is required fold coverage = \[\log(-1/(E - 1))\]
get_summary (self)
    Return a summary (dictionary) for the current fold coverage

get_table (self, 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

3.11.7 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.

    Pre-defined lists are available from ENA. We refer to them as virus, plasmid, phage, archaealvirus, archaea, bacteria, organelle, viroid. In addition we have predefined lists within Sequana. For now, there is one named macaca_fascicularis.

    Warning: the header of the FASTA files are changed to add the GI number instead of embl.

constructor

download_accession (self, acc, output='Custom')
    Download a specific FASTA file given its ENA accession number

download_archaea (self)

download_archaealvirus (self)

download_bacteria (self)
    organisms (may 2016)

    Note: this download method is the longest to end. It took about 20mins on a good connection.


download_fasta (self, filelist, output_dir=None, from_ena=True)
    Download a FASTA (or list of)

    Parameters: filelist – a name to find on the ENA web server OR the name of an accession number.

    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.


download_list (self)
    Download all standard lists of accession numbers from ENA

download_macaca (self)

download_organelle (self)
download_phage(self)
download_plasmids(self)
download_viroid(self)
download_viruses(self)

ena_id_to_gi_number(self, identifiers)

switch_header_to_gi(self, acc)

Kraken will only accept the GI from NCBI so we need to convert the ENA accession to GI numbers

class EUtilsTools
Simple wrapper around EUtils to fetch basic information about an accession number

```
>>> from sequana.databases import 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(self, ids)
An accession or list of them returns list of dictionaries

### 3.11.8 Experimental design

Module to handle experimental design files (adapters)

Sequencers or experimentalists create so-called design files to store information about the sequencing experiment. For example the name of the samples, the sample well, and the index sequences.

The format used to store the design information may vary from one sequencing platform to another. The design file can be used for many different purposes. Currently, we only use them to retrieve information about adapters.

Since there are different formats possible, we decided on a minimal and common set of information. For now, this is a CSV file with the following minimal header:

```
Sample_ID, Index1_Seq
```

or, for double-indexing:

```
Sample_ID, Index1_Seq, Index2_Seq
```

Users should only use the `ExpDesignAdapter` class, which understands the different formats. Currently, the design file created by MiSeq Illumina machine

```
class ExpDesignAdapter (filename, verbose=True)
Generic Experimental design class

This class is used to store the mapping between sample ID and adapter used.

The design information is stored as a dataframe in the attribute df.

The input format is not unique. There are currently 3 different inputs possible as defined in

* ExpDesignGeneric
* ExpDesignMiSeq
```

3.11. References
• **ExpDesignHiSeq** (2500)

The dataframe index is the list of sample identifiers (Sample_ID). The columns contain at least the following:

| Index1_Seq, Index1_ID, Index2_Seq, Index2_ID |

Example:

```python
from sequana import *
filename = sequana_data('test_test_expdesign_hiseq.csv')
eda = ExpDesignAdapter(filename)
```

**constructor**

**Parameters**

- **filename** *(str)* – the input design file. Can also be an instance of `ExpDesignAdapter` itself.
- **verbose** *(bool)* –

**class ExpDesignMiSeq (filename)**

Dedicated experimental design format from Illumina MiSeq sequencers

This MiSeq design format has the following format:

```plaintext
[Header]
blabla

[Reads]
blabla

[Settings]
blabla

[Data]
blabla
```

In the Data section, a CSV file is to be found with the following header:

```
Sample_ID,Sample_Name,Sample_Plate,Sample_Well,I7_Index_ID,index,
Sample_Project,Description
```

The index column may be prefixed. For instance as NFXX where XX is the index so NF should be dropped.

If double-indexing, the header is:

```
Sample_ID,Sample_Name,Sample_Plate,Sample_Well,I7_Index_ID,index,I5_Index_ID,index2,Sample_Project,Description
```

```python
filename = sequana_data("test_expdesign_miseq_illumina_1.csv")
ff = ExpDesignMiSeq(filename)
ff.df
```

**class ExpDesignHiSeq (filename, sep='\',)**

Dedicated experimental design format created by a demultiplexing soft.

This format is used by a demultiplex software used locally at biomics platform. The format of the header is:
FCID, Lane, SampleID, SampleRef, Index Seq, Description, Control, Recipe, Operator, →Project

This is a format that may change in the future.
The SampleID is converted into Sample_ID, “Index Seq”. Note that “Index Seq” may be empty, or filled with an index sequence, or 2 index sequences separated by a “-” sign.

note also FCID = flowcell ID

**class ExpDesignBase (filename, sep=',')**
The Base class for all ExpDesignAdapter classes

The input filename must be a CSV file with at least the following column in the header:

| Sample_ID |

Derived class must define at least **Index1_Seq** and possibly **Index2_Seq**.

Examples of specialised classes are **ExpDesignMiSeq, ExpDesignHiSeq**.

**check (self)**
Check the presence of the Sample_ID column

**read (self)**
Read a CSV file

### 3.11.9 FASTQ module

Utilities to manipulate FASTQ and Reads

**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 = Identifier('@HWUS1-EAS100R:6:73:941:1973#0/1')
>>> ident.info['x_coordinate']
'15343'
```

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]
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 (self)
  Return number of lines

  count_reads (self)
  Return count_lines divided by 4

  extract_head (self, 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 (self, 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 (self)

joining (self, 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 (self)

rewind (self)
Allows to iter from the beginning without opening the file or creating a new instance.

select_random_reads (self, 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)
```

split_chunks (self, N=10)
Not implemented

split_lines (self, N=100000, gzip=True)
Not implemented

stats (self)

to_fasta (self, output_filename='test.fasta')
Slow but works for now in pure python with input compressed data.

to_kmer_content (self, 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.

to_krona (self, 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:
class FastQC (filename, max_sample=500000, dotile=False, verbose=True)

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", "testing")
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` *(self, hold=False, ax=None)*

Boxplot quality

Same plots as in FastQC that is average quality for all bases. In addition a 1 sigma error enveloppe is shown (yellow).

Background separate zone of good, average and bad quality (arbitrary).

`get_actg_content` *(self)*

`get_stats` *(self)*

`histogram_gc_content` *(self)*

Plot histogram of GC content

```python
from sequana import sequana_data
from sequana import FastQC
filename = sequana_data("test.fastq", "testing")
qc = FastQC(filename)
qc.histogram_gc_content()
```

`histogram_sequence_coordinates` *(self)*

Histogram of the sequence coordinates on the plate

```python
from sequana import sequana_data
from sequana import FastQC
filename = sequana_data("test.fastq", "testing")
qc = FastQC(filename)
qc.histogram_sequence_coordinates()
```

**Note:** in this data set all points have the same coordinates.

`histogram_sequence_lengths` *(self, logy=True)*

Histogram sequence lengths

```python
from sequana import sequana_data
from sequana import FastQC
filename = sequana_data("test.fastq", "testing")
qc = FastQC(filename)
qc.histogram_sequence_lengths()
```

`imshow_qualities` *(self)*

Qualities

```python
from sequana import sequana_data
from sequana import FastQC
```
```python
filename = sequana_data("test.fastq", "testing")
qc = FastQC(filename)
qc.imshow_qualities()
from pylab import tight_layout; tight_layout()
```

```python
plot_acgt_content(self, stacked=False)
Plot histogram of GC content
```

```python
from sequana import sequana_data
from sequana import FastQC
filename = sequana_data("test.fastq", "testing")
qc = FastQC(filename)
qc.plot_acgt_content()
```

```python
is_fastq(filename)
```
3.11.10 FASTA module

Utilities to manipulate FASTQ and Reads

```python
class FastA(filename, verbose=False)
    Class to handle FastA files. Cannot be compressed

    property comments
    format_contigs_denovo(self, output_file, len_min=500)
        Replace NODE with the project name and remove contigs with a length lower than len_min.

        Parameters
        • output_file (str) – output file name.
        • len_min (int) – minimal length of contigs.

        Example:
        from sequana import FastA
        contigs = FastA(“denovo_assembly.fasta”) contigs.format_contigs_denovo(“path/to/file.fasta”, len_min=500)
        Results are stored in “path/to/file.fasta”.

get_lengths_as_dict(self)

get_stats(self)

    property lengths

    property names

    next(self)

    reverse_and_save(self, filename)

    save_ctg_to_fasta(self, ctgname, outname)

    select_random_reads(self, 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

    to_fasta(self, 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(self, output)
```

3.11. References
3.11.11 Sequence module

```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
>>> d = DNA("ACGTTTT")
>>> d.complement
>>> d.reverse_complement
```

Some long computations are done when setting the window size:

```python
d.window = 100
```

The ORF detection has been validated against 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.

**Todo:** use counter only once as a property

```python
property AT_skew
property GC_skew
property ORF_pos
barplot_count_ORF_CDS_by_frame(self, alpha=0.5, bins=40, xlabel='Frame', ylabel='\#', bar_width=0.35)
```

```python
entropy(self, sequence)
```

```python
hist_ORF_CDS_linearscale(self, alpha=0.5, bins=40, xlabel='Length', ylabel='\#')
```

```python
hist_ORF_CDS_logscale(self, alpha=0.5, bins=40, xlabel='Length', ylabel='\#')
```

```python
plot_all_skews(self, figsize=(10, 12), fontsize=16, alpha=0.5)
```

```python
property threshold
property type_filter
property type_window
property window
```

```python
class RNA(sequence)
```

Simple RNA class
>>> d = RNA("ACGUUUU")
>>> d.complement
>>> 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**(self, 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**

**property threshold**

**class Sequence** (sequence, complement_in=b'ACGT', complement_out=b'TGCA', letters='ACGT')

Abstract base classe for other specialised sequences such as DNA.

Sequenced is the base class for other classes such as DNA and RNA.

```python
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
Sequana, Release 0.8.0

```python
check(self)
    Check that all letters are valid

complement(self)
    Alias to get_complement()

gc_content(self)
    Return mean GC content

get_complement(self)
    Return complement

get_occurences(self, pattern, overlap=False)
    Return position of the input pattern in the sequence

>>> from sequana import Sequence
>>> s = Sequence('ACGTTTTACGT')
>>> s.get_occurences("ACGT")
[0, 7]

get_reverse(self)
    Return reverse sequence

get_reverse_complement(self)
    Return reverse complement

reverse(self)
    Alias to get_reverse()

reverse_complement(self)
    Alias to get_reverse_complement

property sequence
    Return basic stats about the number of letters

3.11.12 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

3.11.13 ITOL module

class ITOL(tree)
    Tree with branch lengths:

    (A:0.1,(B:0.1,C:0.1)));

    Tree with bootstrap and branch lengths:
```

3.11. References
add_file(self, filename)
export(self, filename='test.png')
upload(self)

3.11.14 IOTools module

class YamlDocParser(filename)
A simple parser to extract block content to be found in YAML files
So as to create tooltips automatically in Sequanix: GUI for snakemake workflows, one can comment YAML configuration file with block comments (see developers guide in Developer guide)
Once read and parsed, all block comments before top-level sections are to be found in the dictionary sections.

```python
from sequana import snaketools
from sequana.iotools import YamlDocParser
module = snaketools.Module('quality_control')
r = YamlDocParser(module.config)
r.sections['fastqc']
```

Those lines are removed from the docstring but available as a dictionary
constructor

Parameters filename (str) – the YAML file to parse

```python
# main documentation
# block comment
section1:
    - item
# block comment
section2:
# a comment
section3:
```

Here, section1 and section2 have block comments but not section3

### 3.11.15 Taxonomy related (Kraken - Krona)

class KrakenResults (filename='kraken.out')

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.

```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
- the next k-mer was not in the database
- the last 3 k-mers mapped to taxonomy ID #562

See kraken documentation for details.

**Note:** a taxon of ID 1 (root) means that the read is classified but in different domain. [https://github.com/DerrickWood/kraken/issue/100](https://github.com/DerrickWood/kraken/issues/100)

**Note:** This takes care of fetching taxons and the corresponding lineages from online web services.

---

### 3.11. References

121
**constructor**

Parameters **filename** – the input from KrakenAnalysis class

**boxplot_classified_vs_read_length**(self)

Show distribution of the read length grouped by classified or not

**property df**

**get_taxonomy_db**(self, **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

**kraken_to_csv**(self, **filename**, **dbname**)

**kraken_to_json**(self, **filename**, **dbname**)

**kraken_to_krona**(self, **output_filename=None**, **mode=None**, **nofile=False**)

**Returns** status: True is everything went fine otherwise False

**plot**(self, **kind='pie'**, **cmap='tab20c'**, **threshold=1**, **radius=0.9**, **textcolor='red'**, ****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("test_kraken.out", "testing")
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

**property taxons**

**to_js**(self, **output='krona.html'**, **onweb=False**)

**class KrakenPipeline**(fastq, database, threads=4, output_directory='kraken', dbname=None)

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.
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/ 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(self, 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(self)
```
Opens the filename defined in the constructor

class KrakenAnalysis(fastq, database, threads=4)
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):

```python
from sequana import KrakenDownload
kd = KrakenDownload()
kf.download_kraken_toydb()
```

See also:
KrakenDownload for more database and sequana.kraken_builder.KrakenBuilder to build your own 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:

```python
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:

```python
from sequana import sequana_data
data = sequana_data("Hm2_GTGAAA_L005_R1_001.fastq.gz", "data")
ka = KrakenAnalysis(data, database=database)
ka.run()
```

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
- `output` – output filename of the Krona HTML page
- `return` –

```python
run(self, 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 KrakenDownload

Utility to download Kraken DB and place them in a local directory:

```python
from sequana import KrakenDownload
kd = KrakenDownload()
kd.download('toydb')
kd.download('minikraken')
```

A large database (8Gb) is available on synapse and has the following DOI:

```plaintext
doi:10.7303/syn6171000
```

It can be downloaded manually or if you have a Synapse login (https://www.synapse.org), you can use:

```python
from sequana import KrakenDownload
kd = KrakenDownload()
kd.downloaded("sequana_db1")
```
download (self, name, verbose=True)

dv = <easydev.tools.DevTools object>

class KrakenHierarchical (filename_fastq, fof_databases, threads=1, output_directory='./kraken_hierarchical/', keep_temp_files=False, output_filename_unclassified=None, force=False)

Kraken Hierarchical Analysis

This runs Kraken on a FastQ file with multiple k-mer databases in a hierarchical 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.

**run** (self, dbname='multiple', output_prefix='kraken_final')

Run the hierarchical analysis

This method does not return anything but 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.

class MultiKrakenResults (filenames, sample_names=None)

def get_df (self)

def plot_stacked_hist (self, output_filename=None, dpi=200, kind='barh', fontsize=10, edge_color='k', lw=1, width=1, ytick_fontsize=10)

class KronaMerger (filename)

Utility to merge two Krona files

Imagine those two files (formatted for Krona; first column is a counter):
You can merge the two files. The first and last lines correspond to the same taxon (species1) so we should end up with a new Krona file with 4 lines only.

The test files are available within Sequana as test_krona_k1.tsv and test_krona_k2.tsv:

```python
from sequana import KronaMerger, sequana_data
k1 = KronaMerger(sequana_data("test_krona_k1.tsv"))
k2 = KronaMerger(sequana_data("test_krona_k2.tsv"))
k1 += k2
# Save the results. Note that it must be tabulated for Krona external usage
k1.to_tsv("new.tsv")
```

**Warning:** separator must be tabulars

### constructor

**Parameters** `filename` *(str)* —

**to_tsv** *(self, output_filename)*

Save the content into a new file in TSV format

### class KrakenBuilder *(dbname)*

This class will help you building a custom Kraken database

You will need a few steps, and depending on the FASTA files you want to include lots of resources (memory and space wise). In the following example, we will be reasonable and use only viruses FASTA files.

First, we need to create the data structure directory. Let us call it `virusdb`:

```python
from sequana import KrakenBuilder
kb = KrakenBuilder("virusdb")
```

We then need to download a large taxonomic database from NCBI. You may already have a local copy, in which case you would need to copy it in virusdb/taxonomy directory. If not, type:

```python
kb.download_taxonomy()
```

The virusdb/taxonomy directory will contain about 8.5G of data.

Note that this currently requires the unix tools `wget` and `tar`.

Then, we need to add some fasta files. You may download specific FASTA files if you know the accession numbers using `download_accession()`. However, we also provide a method to download all viruses from ENA:

```python
kb.download_viruses()
```

This will take a while to download the more than 4500 FASTA files (10 minutes on a good connection). You will end up with a data set of about 100 Mb of FASTA files.

### 3.11. References

---

<table>
<thead>
<tr>
<th>1401</th>
<th>Bacteria</th>
<th>Proteobacteria</th>
<th>species1</th>
</tr>
</thead>
<tbody>
<tr>
<td>591</td>
<td>Bacteria</td>
<td>Proteobacteria</td>
<td>species4</td>
</tr>
<tr>
<td>184</td>
<td>Bacteria</td>
<td>Proteobacteria</td>
<td>species3</td>
</tr>
<tr>
<td>132</td>
<td>Bacteria</td>
<td>Proteobacteria</td>
<td>species2</td>
</tr>
<tr>
<td>32</td>
<td>Bacteria</td>
<td>Proteobacteria</td>
<td>species1</td>
</tr>
</tbody>
</table>
If you wish to download other FASTA (e.g. all bacteria), you will need to use another class from the sequana.

```python
from sequana.databases import ENADownload
ena = ENADownload()
ena.download_fasta("bacteria.txt", output_dir="virusdb/library/added")
```

Please see the documentation for more options and list of species to download.

It is now time to build the DB itself. This is based on the kraken tool. You may do it yourself in a shell:

```
kraken-build --rebuild -db virusdb --minimizer-len 10 --max-db-size 4 --threads 4
--kmer-len 26 --jellyfish-hash-size 500000000
```

Or you the KrakenBuilder. First you need to look at the params attribute. The most important key/value that affect the size of the DB are:

```python
kb.params['kmer_length'] (max value is 31)
kb.params['max_db_size'] is the max size of the DB files in Gb
kb.params['minimizer_len']
```

To create a small DB quickly, we set those values:

```python
kb.params['kmer_length'] = 26
kb.params['minimizer_len'] = 10
```

However, for production, we would recommend 31 and 13 (default)

This takes about 2 minutes to build and the final DB is about 800Mb.

Lots of useless files are in the directory and can be removed using kraken itself. However we do a little bit more and therefore have our own cleaning function:

```
kb.clean_db()
```

Kraken-build uses jellyfish. The hash_size parameter is the jellyfish hash_size parameter. If you set it to 6400M, the memory required is about 6.9bytes times 6400M that is 40Gb of memory. The default value used here means 3.5Gb are required.

The size to store the DB itself should be

\[
\text{Math } sD + 8 (4^M)
\]

where \( s \) is about 12 bytes (used to store a kmer/taxon pair, \( D \) is the number of kmer in the final database, which cannot be estimated before hand, and \( M \) the length minimiser parameter.

**Constructor**

**Parameters**

- **dbname (str)** – Create the Kraken DB in this directory

- **clean_db (self)**
  - Once called, you will not be able to append more FASTA files

- **download_accession (self, acc)**
  - Donwload a specific Fasta from ENA given its accession number

  Note that if you want to add specific FASTA from ENA, you must use that function to make sure the header will be understood by Kraken; The header must use a GI number (not ENA)
download_ncbi_refseq(self, category)
    Download all files of type fna from ncbi FTP.

    kb = KrakenBuilder()
    kb.download_ncbi_refseq("viral")

download_taxonomy(self, force=False)
    Download kraken data

    The downloaded file is large (1.3Gb) and the unzipped file is about 9Gb.

    If already present, do not download the file except if the force parameter is set to True.
download_viruses(self)
get_gis(self, extensions=['fa'])
get_taxons_from_gis(self, gis, filename='gi_taxid_nucl.dmp')
init(self)
run(self, dbs=[], download_taxon=True)
    Create the Custom Kraken DB
    1. download taxonomy files
    2. Load the DBs (e.g. viruses)
    3. Build DB with kraken-build
    4. Clean it up

3.11.16 Pacbio module

Pacbio QC and stats
class PacbioMappedBAM(filename, method)
    Parameters filename (str) – input BAM file

    boxplot_mapq_concordance(self)
filter_mapq(self, 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(self, reference_length=None)

hist_GC(self, 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) –
Sequana, Release 0.8.0

- **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()
```

![Histogram of GC%](image)

**Mean GC : 62.46**

**hist_concordance** *(self, bins=100, fontsize=16)*

 Formula: $1 - \left( \frac{\text{in} + \text{del} + \text{mismatch}}{\text{in} + \text{del} + \text{mismatch} + \text{match}} \right)$

For BWA and BLASR, the get_cigar_stats are different !!! BWA for instance has no X stored while Paebio forbids the use of M (CMATCH) tag. Instead, it uses X (CDIFF) and = (CEQUAL) characters.

**Subread Accuracy:** The post-mapping accuracy of the basecalls. Formula: $[1 - \left( \frac{\text{errors}}{\text{subread length}} \right)]$, where errors = number of deletions + insertions + substitutions.

**hist_median_ccs** *(self, bins=1000, **kwargs)*

Group subreads by ZMW and plot median of read length for each polymerase

Chapter 3. User guide and reference
**hist_read_length** (self, bins=80, alpha=0.5, 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(self, 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 (self)

to_fasta (self, 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

```python
to_fastq(self, 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:

```
qStart qEnd
0 | aStart aEnd | [–—-…—-———––…——] < “ZMW read” coord. system
   ~~~~~~~~~~~~~~~~~~~~ < query; “-” =aligning subseq.
   [–—-…—-———––…——] < “ref.” / “target” coord. system 0 tStart tEnd
```

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)

property of

**filter_length** *(self, 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** *(self, min_length=50, max_length=15000)*

**get_number_of_ccs** *(self, min_length=50, max_length=15000)*

**hist_GC** *(self, 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** *(self, bins=None, alpha=0.5, hold=False, fontsize=12, grid=True, xlabel='Number of ZMW passes', logy=True, ylabel='#', label='', title='Number of ZMW passes')*
Plot histogram of number of reads per ZMW (number of passes)

Parameters

- **alpha** *(float)* – transparency of the histograms
- **hold** *(bool)* –
Mean GC : 62.46
Sequana, Release 0.8.0

- `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()
```

![Number of ZMW passes](image)

`hist_read_length (self, bins=80, alpha=0.5, 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
from sequana.pacbio import PacbioSubreads
from sequana import sequana_data
b = PacbioSubreads(sequana_data("test_pacbio_subreads.bam"))
b.hist_read_length()
• **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**(self, 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**

```python
random_selection(self, 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 original 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
class PBSim(input_bam, simul_bam)

Filter an input BAM (simulated with pbsim) so as so keep reads that fit a target distribution.
This uses a MH algorithm behind the scene.

```python
ss = pacbio.PBSim("test10X.bam")
clf();
ss.run(bins=100, step=50)
```

For example, to simulate data set, use:

```bash
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

```python
run (self, bins=50, xmin=0, xmax=30000, step=1000, burn=1000, alpha=1, output_filename=None)
```

data_distribution (self, xprime)

The target distribution

Compute histogram. Get X, Y. Given xprime, interpolate to get yprime use e.g. np.interp

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** (self, 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** (self, 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** (self, 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_read_length(self, bins=80, alpha=0.5, hold=False, fontsize=12, grid=True, xlabel='Read Length', ylabel='#', label='', title=None, logy=False, ec='k', hist_kwgs={} )

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)** —
from sequana.pacbio import PacbioSubreads
from sequana import sequana_data
b = PacbioSubreads(sequana_data("test_pacbio_subreads.bam"))
b.hist_read_length()

Read length
Mean length : 1405.68

plot_GC_read_len(self, 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) -

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])
GC % vs length
Mean length: 1405.68, Mean GC: 62.46

Read length
GC %

Mean length: 1405.68, Mean GC: 62.46
Sequana, Release 0.8.0

reset (self)

to_fasta (self, 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 (self, 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 (self, bins=10, fontsize=12)

  hist_polymerase_per_barcode (self, bins=10, fontsize=12)
     histogram of number of polymerase per barcode
     Cumulative histogram gives total number of polymerase reads

  hist_quality_per_barcode (self, bins=10, fontsize=12)

  plot_and_save_all (self, dpi=100, directory='.')

  plot_polymerase_per_barcode (self, fontsize=12, unbarcoded=True)
     Number Of Polymerase Reads Per Barcode

  plot_subreads_histogram (self, bins=10, fontsize=12)

3.11.17 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/

3.11. References

145
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('BCCFFFFFHAAAAHIIJJJJJJJJJJJJJJFH')
q.plot()
q.offset = 64
q.plot()

from pylab import legend
legend(loc="best")
```

```latex
\begin{figure}
\centering
\includegraphics[width=\textwidth]{example.png}
\caption{Quality per base for different offsets.}
\end{figure}
```

```python
class Quality(seq, offset=33):
    Phred quality

>>> from sequana.phred import Quality
>>> q = Quality('BCCFFFFFHAAAAHIIJJJJJJJJJJJJJJFH')
>>> q.plot()
```

You can access to the quality as a list using the `quality` attribute and the mean quality from the `mean_quality` attribute.
Sequana, Release 0.8.0

```python
property mean_quality
    return mean quality

plot(self, 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
    • a low proba of errors (0.05) gives Q = 13
    • a low proba of errors (0.01) gives Q = 20

quality_to_proba_sanger(quality)
    Quality to probability (Sanger)
```

### 3.11.18 Running median

Data analysis tool

<table>
<thead>
<tr>
<th>RunningMedian(data, width[, container])</th>
<th>Running median (fast)</th>
</tr>
</thead>
</table>

```python
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](http://code.activestate.com/recipes/576930/#c3) that uses a simple list as proposed in [https://gist.github.com/f0k/2f8402e4db6974bfef1](https://gist.github.com/f0k/2f8402e4db6974bfef1) and was adapted to our needs included object oriented implementation.
```

**Note:** a circular running median is implemented in `sequana.bedtools.GenomeCov`

```python
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

```python
from scipy import *
import scipy.signal
(continues on next page)```
```python
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()
```

```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")
```

(continues on next page)
**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(self)
```

```python
running_median(data, width, container=list)
```

### 3.11.19 Snakemake module

Set of tools to manipulate Snakefile and config files

Here is an overview (see details here below)

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sequana.snaketools.DOTParser</code></td>
<td>Utility to manipulate the dot file returned by Snakemake</td>
</tr>
<tr>
<td><code>sequana.snaketools.FastQFactory</code></td>
<td>FastQ Factory tool</td>
</tr>
<tr>
<td><code>sequana.snaketools.FileFactory</code></td>
<td>Factory to handle a set of files</td>
</tr>
<tr>
<td><code>sequana.snaketools.Module</code></td>
<td>Data structure that holds metadata about a Module</td>
</tr>
<tr>
<td><code>sequana.snaketools.ModuleFinderSingleton</code></td>
<td>Data structure to hold the Module names</td>
</tr>
<tr>
<td><code>sequana.snaketools.PipelineManager</code></td>
<td>Utility to manage easily the snakemake pipeline</td>
</tr>
<tr>
<td><code>sequana.snaketools.SnakeMakeStats</code></td>
<td>Interpret the snakemake stats file</td>
</tr>
<tr>
<td><code>sequana.snaketools.SequanaConfig</code></td>
<td>Reads YAML config file and ease access to its contents</td>
</tr>
<tr>
<td><code>sequana.snaketools.message</code></td>
<td>Dedicated print function to include in Snakefiles</td>
</tr>
<tr>
<td><code>sequana.snaketools.modules</code></td>
<td></td>
</tr>
</tbody>
</table>

#### class DOTParser (filename, mode='v2')

Utility to manipulate the dot file returned by Snakemake

This class is used in the *dag* and *rulegraph* rules used in the snakemake pipeline. The input must be a dag/rulegraph created by snakemake.

Consider this example where the test file was created by snakemake –dag

```python
from sequana import sequana_data
from sequana.snaketools import DOTParser

filename = sequana_data("test_dag.dot")
dot = DOTParser(filename)

# creates test_dag.ann.dot locally
dot.add_urls("test.dot", {"fastqc": "fastqc.html"})

You can then convert the dag in an unix shell:

```bash
dot -Tsvg test.ann.dot -o test.svg
```

### 3.11. References

151
constructor

Parameters filename (str) – a DAG in dot format created by snakemake

add_urls (self, output_filename=None, mapper={}, title=None)
Change the dot file adding URL on some nodes

Parameters

• output_filename (str) – the DAG file in dot format (graphviz)
• mapper (dict) – a dictionary where keys are named after the rule names for which an HTML will be available (provided here as keys)

class FastQFactory (pattern, extension=['fq.gz', 'fastq.gz'], read_tag='_R[12]_', verbose=False, paired=True)

FastQ Factory tool

In NGS experiments, reads are stored in a so-called FastQ file. The file is named:

PREFIX_R1_SUFFIX.fastq.gz

where _R1_ tag is always to be found. This is a single-ended case. In paired case, a second file is to be found:

PREFIX_R2_SUFFIX.fastq.gz

The PREFIX indicates the sample name. The SUFFIX does not convey any information per se. The default read tag ("_R[12]_*") handle this case. It can be changed if data have another read tags. (e.g. "[12].fastq.gz")

Yet, in long reads experiments (for instance), naming convention is different and may nor be single/paired end convention.

In a directory (recursively or not), there could be lots of samples. This class can be used to get all the sample prefix in the tags attribute.

Given a tag, one can get the corresponding file(s):

```python
ff = FastQFactory("*fastq.gz")
ff.tags
ff.get_file1(ff.tags[0])
len(ff)
```

Constructor

Parameters

• pattern (str) – a global pattern (e.g., H*fastq.gz)
• extension (list) – not used
• read_tag (str) – regex tag used to join paired end files. Some characters need to be escaped with a ‘’ to be interpreted as character. (e.g. ”_R[12]_.fastq.gz”)
• verbose (bool) –

get_file1 (self, tag=None)
get_file2 (self, tag=None)
property paired
  guess whether data is paired or not
class FileFactory(pattern)
    Factory to handle a set of files

    from sequana.snaketools import FileFactory
    ff = FileFactory("H*.gz")
    ff.filenames

    A set of useful methods are available based on this convention:

    >>> fullpath = /home/user/test/A.fastq.gz
    >>> dirname(fullpath)
    '/home/user/test'
    >>> basename(fullpath)
    'A.fastq.gz'
    >>> realpath(fullpath)  # is .., expanded to /home/user/test

    >>> all_extensions
    "fastq.gz"
    >>> extensions
    ".gz"

    FIXME: pathname to be checked.

    A basename is the name of a directory in a Unix pathname that occurs after the last slash.
    dirname, returns everything but the final basename in a pathname. Both
    The pathname is a specific label for a file’s directory location while within an operating system.

    Constructor

    Parameters pattern – can be a filename, list of filenames, or a global pattern (a unix regular
    expression with wildcards). For instance, "*/fastq.gz"

    Warning: Only in Python 3.X supports the recursive global pattern for now.

    property all_extensions
    the extensions (list)

    property basenames
    list of filenames and their extensions without the path

    property extensions
    the last extension (list)

    property filenames
    list of filenames (no path, no extension)

    property pathname
    the common relative path

    property pathnames
    the relative path for each file (list)

    property realpaths
    real path is the full path + the filename the extension
**class Module**(name)

Data structure that holds metadata about a Module

In Sequana, we provide rules and pipelines to be used with snakemake. Snakemake rules look like:

```
rule <name>:
    :input: file1
    :output: file2
    :shell: "cp file1 file2"
```

A pipeline may look like:

```
include: "path_to_rule1"
include: "path_to_rule2"
rule all:
    input: FINAL_FILES
```

Note that the pipeline includes rules by providing the path to them.

All rules can be stored in a single directory. Similarly for pipelines. We decided not to use that convention. Instead, we bundle rules (and pipelines) in their own directories so that other files can be stored with them. We also consider that

1. if the **Snakefile** includes other **Snakefile** then it is **Pipeline**.
2. Otherwise it is a simple **Rule**.

So, a Module in sequana’s parlance is a directory that contains a rule or a pipeline and associated files. There is currently no strict conventions for rule Modules except for their own rule file. However, pipeline Modules should have the following files:

- A **snakemake** file named after the directory with the extension **.rules**
- A **README.rst** file in restructured text format
- An optional config file in YAML format named config.yaml. Although json format is possible, we use YAML throughout sequana for consistency. Rules do not have any but pipelines do. So if a pipeline does not provide a config.yaml, the one found in ./sequana/sequana/pipelines will be used.
- a **requirements.txt**

**Note:** Developers who wish to include new rules should refer to the Developer guide.

**Note:** it is important that module’s name should be used to name the directory and the rule/pipeline.

The Modules are stored in sequana/rules and sequana/pipelines directories. The modules’ names cannot be duplicated.

Example:

```
pipelines/test_pipe/test_pipe.rules
pipelines/test_pipe/README.rst
rules/rule1/rule1.rules
rules/rule1/README.rst
```

The Module will ease the retrieval of information linked to a rule or pipeline. For instance if a pipeline has a config file, its path can be retrieved easily:
This Module may be rule or pipeline, the method `is_pipeline()` can be used to get that information.

**Constructor**

**Parameters** `name (str)` – the name of an available module.

**check (self, mode='warning')**

**property cluster_config**
full path to the config cluster file of the module

**property config**
full path to the config file of the module

**property description**
Content of the README file associated with

**is_executable (self, verbose=False)**
Is the module executable
A Pipeline Module should have a requirements.txt file that is introspected to check if all executables are available;

**Parameters** `verbose` –

**Returns** a tuple. First element is a boolean to tell if it executable. Second element is the list of missing executables.

**is_pipeline (self)**
Return true is this module is a pipeline

**property logo**
full path to the logo of the module

**md5 (self)**
return md5 of snakefile and its default configuration file

```python
>>> from sequana import snaketools as sm
>>> m = sm.Module("variant_calling")
>>> m.md5()
{'config': 'e23b26a2ff45fa9ddb36c40670a8a00e',
'snakefile': '7d3917743a6b123d9861ddbb5f3baef'}
```

**property multiqc_config**
full path to the multiqc config file of the module

**property name**
name of the module

**property overview**

**property path**
full path to the module directory

**property readme**
full path to the README file of the module
property requirements
  list of requirements

property schema_config
  full path to the schema config file of the module

property snakefile
  full path to the Snakefile file of the module

property version
  Get version

class PipelineManager (name, config, pattern="*.fastq.gz", fastq=True)
Utility to manage easily the snakemake pipeline

Inside a snakefile, use it as follows:

```python
from sequana import PipelineManager
manager = PipelineManager("pipeline_name", "config.yaml")
```

config file must have these fields:

- input_directory: #a_path
- input_readtag: _R[12]_ # default
- input_pattern: # a_global_pattern e.g. H*fastq.gz

The manager can then easily access to the data with a `FastQFactory` instance:

```python
manager.ff.filenames
```

This can be further used to get a wildcards with the proper directory.

The manager also tells you if the samples are paired or not assuming all samples are homogneous (either all paired or all single-ended).

If there is only one sample, the attribute `mode` is set to “nowc” meaning no wildcard. Otherwise, we assume that we are in a wildcard mode.

When the mode is set, two attributes are also set: `sample` and `basename`.

If the mode is `nowc`, the `sample` and `basename` are hardcoded to the sample name and sample/rule/sample respectively. Whereas in the `wc` mode, the sample and basename are wildcard set to “{sample}” and “{sample}/rulename/{sample}”. See the following methods `getname()`.

For developers: the config attribute should be used as getter only.

**Constructor**

**Parameters**

- **name** – name of the pipeline
- **config** – name of a configuration file
- **pattern** – a default pattern if not provided in the configuration file as an `input_pattern` field.

**error** *(self, msg)*

**getlogdir** *(self, rulename)*
Create log directory: */sample/logs/sample_rule.logs

3.11. References
**getname** *(self, rulename, suffix=None)*  
In the basename, include rulename and suffix

**getrawdata** *(self)*  
Return list of raw data

If `mode` is `nowc`, a list of files is returned (one or two files) otherwise, a function compatible with `snakemake` is returned. This function contains a wildcard to each of the samples found by the manager.

**getreportdir** *(self, acronym)*  
Create the report directory.

**getwkdir** *(self, rulename)*

**message** *(self, msg)*

**property paired**

**class PipelineManagerGeneric** *(name, config, sample_func=None)*  
For all files except FastQ, please use this class instead of `PipelineManager`.

**getlogdir** *(self, rulename)*  
Create log directory: */sample/logs/sample_rule.logs

**getname** *(self, rulename, suffix=None)*  
In the basename, include rulename and suffix

**getrawdata** *(self)*  
Return list of raw data

If `mode` is `nowc`, a list of files is returned (one or two files) otherwise, a function compatible with `snakemake` is returned. This function contains a wildcard to each of the samples found by the manager.

**getreportdir** *(self, acronym)*  
Create the report directory.

**getwkdir** *(self, rulename)*

**class SnakeMakeStats** *(filename, N=1)*  
Interpret the snakemake stats file

Run the Snakemake with this option:

```
-- stats stats.txt
```

Then:

```
from sequana.snaketools import SnakeMakeStats
from sequana import sequana_data
filename = sequana_data("test_snakemake_stats.txt", "testing")
s = SnakeMakeStats(filename)
s.plot()
```
Constructor

```python
plot (self, fontsize=16)
Create the barplot from the stats file

plot_and_save (self, filename='snakemake_stats.png', outputdir='report')
```

```python
class SequanaConfig (data=None, converts_none_to_str=True)
Reads YAML config file and ease access to its contents
This can also be used to check the validity of the config file

```python
>>> sc = SequanaConfig(config)
>>> sc.config.pattern == "*.fastq.gz"
True
```

Empty strings in a config are interpreted as None but SequanaConfig will replace None with empty strings, which is probably what was expected from the user. Similarly, in snakemake when settings the config file, one can override a value with a False but this is interpreted as “False” This will transform back the “True” into True.

Another interest concerns the automatic expansion of the path to directories and files starting with the special ~ (tilde) character, that are expanded transparently.

Could be a JSON or a YAML file

**Parameters**

- **filename** *(str)* – filename to a config file in json or YAML format.

SEQUANA config files must have some specific fields:

##### input_directory

```python
check_config_with_schema (self, schemafile)
Check the config file with respect to a schema file

Sequana pipelines should have a schema file in the Module.
```

```python
cleanup (self)
Remove template elements and change None to empty string.
```

```python
cleanup_config (self)
```

```python
copy_requirements (self, target)
Copy files to run the pipeline

If a requirement file exists, it is copied in the target directory. If not, it can be either an http resources or a sequana resources.
```

```python
save (self, filename='config.yaml', cleanup=True)
Save the yaml code in _yaml_code with comments
```

```python
pipeline_names = ['compressor']
```

list of pipeline names found in the list of modules
3.11.20 Snpeff module

Tools to launch snpEff.

```python
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 creates 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:
```
snpff = SnpEff('file.gbk')
snpff.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.

Constructor

Parameters

• **annotation** – annotation reference.
• **file_format** – format of your file. (‘only genbank actually’)
• **log** – log file
• **snpeff_datadir** –
• **fastafile** – if a GFF is used, you must provide the FASTA input file as well

```python
add_locus_in_fasta(self, 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

```python
launch_snpeff(self, 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.

```python
download_fasta_and_genbank(identifier, tag, genbank=True, fasta=True)
```

Parameters

• **identifier** – valid identifier to retrieve from NCBI (genbank) and ENA (fasta)
• **tag** – name of the filename for the genbank and fasta files.
3.11.21 General tools

misc utilities

textwrap (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)

rest2html (s)
Converts a restructuredText document into HTML.

Note that the returned object is a bytes so need to be decoded with decode()

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

findpos (seq, chr)
Find position(s) of a substring into a longer string.

Note that this function is a generator:

>>> list(findpos("AACCGGAAGGTT", "GG"))
[4, 8]

on_cluster (pattern=[‘tars-‘])
Used to check if we are on a cluster
“tars-” is the name of a cluster’s hostname. Change or append the argument pattern with your cluster’s hostname

Parameters pattern (str) – a list of names (strings) or a string

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

Statistical tools

moving_average (data, n)
Compute moving average
Parameters \( n \) – window’s size (odd or even).

```python
>>> from sequana.stats import moving_average as ma
>>> ma([1,1,1,3,3,3,3,3], 4)
array([ 1. , 1.5, 2. , 2.5, 3. ])
```

Note: the final vector does not have the same size as the input vector.

**evenness** \((data)\)

Return Evenness of the coverage

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

work before or after normalisation but lead to different results.

\[
C = \text{mean}(X) D_2 = X[X <= C] N = \text{len}(X)n = \text{len}(D2) E = 1 - (n - \text{sum}(D2)/C)/N
\]

General tools

```python
class StatsBAM2Mapped() -> new empty dictionary
dict(mapping) -> new dictionary initialized from a mapping object's (key, value) pairs
dict(Iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable: d[k] = v
dict(**kwargs) -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: dict(one=1, two=2)
```

to_html(self)

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 = unmapped reads + R1 mapped + R2 mapped - supplementary reads (those with flag 2048).

```python
class GZLineCounter(filename)
    Fast GZipped line counter
    Uses zcat if possible, otherwise gzip library (twice as slow).

>>> from sequana import sequana_data
>>> from sequana.misc import GZLineCounter
>>> gz = GZLineCounter(sequana_data("test.fastq.gz"))
>>> len(gz)
100
```

### 3.11.22 VCF module

Analysis of VCF file generated by freebayes.

**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** *(self, output_filename, info_field=False)*

Write DataFrame in CSV format.

**Params**

- **str output_filename** output CSV filename.
to_vcf(self, 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.

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,
    "strand_ratio": 0.2
}
filter_v = v.filter_vcf(filter_dict)
filter_v.to_vcf('output.filter.vcf')
```

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.
- kwags – any arguments accepted by vcf.Reader

filter_vcf(self, filter_dict=None)
Filter variants in the VCF file.

Parameters filter_dict (dict) – dictionary of filters. It updates the attribute
VCF_freebayes.filters

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**(self)

**property is_joint**

Get `VCF_freebayes.is_joint` if the vcf file is a joint_freebayes.

**rewind**(self)

Rewind the reader

**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**

**compute_frequency**(vcf_line)

Compute frequency of alternate allele. \( \text{alt_freq} = \frac{\text{Count Alternate Allele}}{\text{Depth}} \)

**Parameters**

- `vcf_line` (vcf.model._Record) – variant record

**compute_strand_balance**(vcf_line)

Compute strand balance of alternate allele include in \([0,0.5]\). \( \text{strand_bal} = \frac{\text{Alt Forward}}{\text{Alt Forward} + \text{Alt Reverse}} \)

**Parameters**

- `vcf_line` (vcf.model._Record) – variant record

FYI: in freebayes, the allele balance (reported under AB), strand bias counts (SRF, SRR, SAF, SAR) and bias estimate (SAP) can be used as well for filtering. Here, we use the strand balance computed as \( \frac{\text{SAF}}{\text{SAF} + \text{SAR}} \)

**strand_ratio**(number1, number2)

Compute ratio between two number. Return result between \([0:0.5]\).
3.11.23 Module Reports

Generic module is the parent module of all other module

```python
class SequanaBaseModule (template_fn='standard.html')
    Generic Module to write HTML reports.
    add_code_section (self, content, language)
        Add code in your html.
    add_float_right (self, content)
        Align a content to right.
    copy_file (self, filename, target_dir)
        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 (self, 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 (self, 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 (self, 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 (self, output_filename)
        Create HTML file with Jinja2.
          Parameters
          • output_filename (str) – HTML output filename
    create_link (self, 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.
```

3.11. References
**Sequana, Release 0.8.0**

```python
include_svg_image(self, filename)
Include SVG image in the html.

png_to_embedded_png(self, png, style=None)
Include a PNG file as embedded file.

required_dir = ('css', 'js', 'images')
```

Report dedicated to BAM file

```python
BAMQCModule(bam_input[, output_filename])
Report dedicated to BAM file

```python
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")
```

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.

```python
add_flag_section(self)
add_images_section(self)
create_report_content(self)
```

Module to write coverage report

```python
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(self)
Create HTML report for each chromosome present in data.

create_chromosome_table(self, html_list)
Create table with links to chromosome reports.

create_report_content(self, html_list)
```
init_roi_datatable(ros)

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

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(self)

basic_stats(self)

Basics statistics section.

coverage_barplot(self)

Coverage barplots section.

coverage_plot(self)

Coverage section.

create_report_content(self, directory, options=None)

Generate the sections list to fill the HTML report.

gc_vs_coverage(self)

3 dimensional plot of GC content versus coverage.

normalized_coverage(self)

Barplot of normalized coverage section.

regions_of_interest(self, rois, links)

Region of interest section.

subcoverage(self, 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:

<table>
<thead>
<tr>
<th>chromosome_name/chromosome_name_0_200000.html</th>
</tr>
</thead>
<tbody>
<tr>
<td>chromosome_name/chromosome_name_200000_400000.html</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>chromosome_name/chromosome_name_2000000_2200000.html</td>
</tr>
<tr>
<td>chromosome_name/chromosome_name_2200000_2300000.html</td>
</tr>
</tbody>
</table>

Note that if the BED file positions does not start at zero, then names will take care of that.

3.11. References
**Barplot of zscore distribution section.**

### 3.11.24 Others

#### data related

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.bam")
```

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:

```python
filenames = sequana_data("*", where="images")
```

Registered directories are:

- data
- testing
- data/adapters
- 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 retrieve 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
fullpath = sequana_data("phiX174.fa", "data")
```

Other files stored in this directory will be documented here.
# report related

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:

    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
    df2 = pandas.read_csv('data2.csv')
    datatable2 = DataTable(df2, 'data2', datatable.datatable)
    html2 = datatable.create_datatable()
```

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 be included in the CSV table

**create_datatable** (**self**, **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** (**self**)  
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 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 I 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/](https://datatables.net/reference/option/)
constructor

Parameters

- **df** - data frame.
- **html_id**(str) – the ID used in the HTML file.

create_javascript_function(self)

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/

**property html_id**

Get the html_id, which cannot be set by the user after the instanciation of the class.

set_links_to_column(self, link_col, target_col)

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(self, 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.

### 3.12 FAQS

#### 3.12.1 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:
conda config --add channels r bioconda
conda install sequana

3.12.2 What are the dependencies

There are two kind 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:

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:

conda install --file https://raw.githubusercontent.com/sequana/sequana/master/requirements_pipelines_extra.txt

3.12.3 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 needs manual installation.

quast

http://quast.bioinf.spbau.ru/manual.html#sec1

wget https://downloads.sourceforge.net/project/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 ...lib/graphviz/libgvplugin_gd.so.6" - file not found
```

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: libbz2.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**

```python
from PyQt5.QtWebKitWidgets import QWebView
...
```

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](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
```

### 3.12.4 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.

### 3.12.5 Sequanix related

For question related to Sequanix, we have a dedicated section in [FAQS](#).

### 3.12.6 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.

### 3.12.7 Variant Calling pipeline

If snpeff fails with this type of errors:

```
java.lang.RuntimeException: Error reading file 'null'
javalo.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:


### 3.12.8 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](https://github.com/sequana/sequana/issues)
3.12.9 Singularity

If you use the singularity container and get this kind of error:

```bash
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:

```bash
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:

```bash
singularity shell sequana-sequana-master.img
```

3.12.10 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:

```ini
backend: tkagg
```

where you can replace tkagg with e.g. qt5agg

3.13 Changelog
* 0.6.4
* 0.6.3.post1
* 0.6.3
* 0.6.2
* 0.6.1
* 0.6.0
* 0.5.2
* 0.5.1
* 0.5.0 August 2017
* 0.4.2 August 2017
* 0.4.1 July 2017
* 0.4 July 2017
* 0.3 April-June 2017
* 0.2 - March - April 2017
* 0.1.21 - Feb 2017
* 0.1.20 - Feb 2017
* 0.1.19 - Feb 2017
* 0.1.17/0.1.18 - Jan 2017

– 2016
* 0.1.16
* 0.1.15
* 0.1.14
* 0.1.13
* 0.1.12
* 0.1.11
* 0.1.10 - July 2016
* 0.1.7 to 0.1.9 - July 2016
* 0.1.6 - June 2016
* prior 0.1.5 June 2016
3.13.1 0.8.0

- Remove all pipelines from sequana. Pipelines have now their own repositories on github to ease the development 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. The compressor pipeline was dropped (redundant with sequana_compressor standalone). Other pipeline have now their own repositories:
  - sequana_coverage
  - sequana_demultiplex
  - sequana_fastqc
  - sequana_revcomp
  - sequana_rnaseq
  - sequana_variant_calling
  - sequana_denovo
  - sequana_pacbio_amplicon
  - new ones
  - sequana_downsampling
  - sequana_mapper

This is an stable release made to includes lots of new features and pipelines and bug fixes made.

- 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_.

---

180 Chapter 3. User guide and reference
– Repeats: for multi fasta with similar header, we were expecting the chrom name to be unique but underly-
ing tool uses regular expression. So, this was buggy when chrom name were starting with same string. e.g
chr1 anc chr11.

– multiqc section of sequana_coverage: duplicate chrom names across multiple samples were shown as a
single entry in the report.

– draft version of multiqc 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 ylimits 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.

  – RNASeq pipeline: removed sartools, kraken. Fixed bamCoverage rule. Simplify usage related to
indexing and mapping. Fixed igttools rule. Fixed the reoderSam rule (wrong executable). Fixed a
incorrect parameter name in bamCoverage rule. Fixed incorrect Snakemake syntax in the fastq_screen
rule and RNAseQC. Fixed another deprecated rule: fastq_screen_report.

  – 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 barcoded 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
3.13.2 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 pachio_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

3.13.3 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.Alignment, 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

3.13.4 0.7.0

- BUGS:
  - add /1 and /2 in quality control pipeline [https://github.com/sequana/sequana/issues/508](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](https://github.com/sequana/sequana/issues/512)
  - Fix reading of SampleSheet for MiSeq: [https://github.com/sequana/sequana/issues/511](https://github.com/sequana/sequana/issues/511)
  - Add Exp Design checked in quality control pipeline: [https://github.com/sequana/sequana/issues/500](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

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](https://github.com/sequana/sequana/issues/501)

0.6.4

- BUGS:
  - Fix issue [https://github.com/sequana/sequana/issues/380](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

3.13. Changelog 183
– add summary module to store summary in json formats.
– simple vcf_filter standalone

• CHANGES:
  – pin kraken version to 1.1 (newest on bioconda)
  – MAJOR REFACTORYING 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 regression bug (https://github.com/sequana/sequana/issues/484)
  – 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)

• 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
  – Set singularity hub (v2.4)

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).
– Sequanix: 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: pacbio_qc finalised
  - pipeline: rnaseq: finalised
  - module pacbio: 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: pacbio_denovo
  - multiqc modules integrated in sequana. See Developer guide for details.
  - module snaketools: 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 yam) - 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:
Sequana, Release 0.8.0

* 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:

    from sequana import BAMReport

is now:

    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
3.13.5 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)
  – snaketools:
    * empty strings are kept as empty strings (not None)
    * remove check() method in SequanaConfig
    * cleanup (removing of templates) ca 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.
  – snaketools:
    * 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
  – Genomcov 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)

3.14 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.
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

3.15 Indices and tables

- genindex
- modindex
- search
a
sequana.adapters, 81
sequana.assembly, 86

b
sequana.bamtools, 90
sequana.bedtools, 92

c
sequana.cigar, 86
sequana.coverage, 101

d
sequana.databases, 104
sequana.datatools, 170

e
sequana.expdesign, 105

f
sequana.fasta, 115
sequana.fastq, 107
sequana.freebayes_vcf_filter, 164

i
sequana.iotools, 120
sequana.itol, 119

k
sequana.kmer, 119
sequana.kraken, 121
sequana.kraken_builder, 127
sequana.krona, 126

m
sequana.misc, 162
sequana.modules_report.bamqc, 168
sequana.modules_report.base_module, 167
sequana.modules_report.coverage, 168

p
sequana.pacbio, 129

sequana.phred, 145

r
sequana.resources.data, 170
sequana.running_median, 148

s
sequana.sequence, 116
sequana.snaketools, 151
sequana.snpeff, 161
sequana.stats, 162

t
sequana.tools, 163

u
sequana.utils.datatables_js, 171
compute_gc_content() (GenomeCov method), 94
count_lines() (FastQ method), 108
count_reads() (Coverage module in sequana.coverage), 101
download_accession() (ENADownload method), 104
download_archaealvirus() (ENADownload method), 104
download_archaea() (ENADownload method), 104
download_accession() (ENADownload method), 125
download_bacteria() (ENADownload method), 104
download_fasta() (ENADownload method), 104
download_fasta_and_genbank() (in module sequana.snpeff), 161
download_list() (ENADownload method), 104
download_macaca() (ENADownload method), 104
download_ncbi_refseq() (KrakenBuilder method), 128
download-organelle() (ENADownload method), 104
download_phage() (ENADownload method), 104
download_plasmids() (ENADownload method), 105
download_taxonomy() (EnaIDtoGiNumber method), 129
download_virus() (ENADownload method), 105
download_viruses() (ENADownload method), 105
download_viroid() (ENADownload method), 105
download_viroids() (KrakenBuilder method), 129

dv (KraekenDownload attribute), 126

df() (BAMSimul property), 141
df() (ChromosomeCov property), 97
df() (DataTable property), 171
df() (FilteredFreebayes property), 164
df() (KrakenResults property), 122
df() (PacBioSubreads property), 134
df_shustring() (Repeats property), 118
DNA (class in sequana.sequence), 116
do_merge() (Repeats property), 118
DOC() (ChromosomeCov property), 97
download_operator() (ENADownload method), 104
download_operator() (ENADownload method), 104
download_accession() (ENADownload method), 104
download_accession() (ENADownload method), 128
download_accession() (KrakenBuilder method), 128
download_accession() (KrakenBuilder method), 128
download_accession() (KrakenBuilder method), 128
download_accession() (KrakenBuilder method), 128
download_accession() (KrakenBuilder method), 128
download_accession() (KrakenBuilder method), 128
download_accession() (KrakenBuilder method), 128
download_accession() (KrakenBuilder method), 128

dv (KraekenDownload attribute), 126

E

ena_id_to_gi_number() (ENADownload method), 105
ENADownload (class in sequana.databases), 104
entropy() (DNA method), 116
error() (PipelineManager method), 157
EUtilsTools (class in sequana.databases), 105
evenness() (ChromosomeCov property), 97
evenness() (in module sequana.stats), 163
ExpDesignAdapter (class in sequana.expdesign), 105
ExpDesignBase (class in sequana.expdesign), 107
ExpDesignHiSeq (class in sequana.expdesign), 106
ExpDesignMiSeq (class in sequana.expdesign), 106
export() (ITOL method), 120
extensions() (FileFactory property), 154
extract_head() (FastQ method), 108

F
FASTA, 194
FASTA (class in sequana.fasta), 115
fasta_fwd_rev_to_columns() (in module sequana.adapters), 85
FastQ (class in sequana.fastq), 108
FastQC (class in sequana.fastq), 110
FastQFactory (class in sequana.snakertools), 153
feature_dict() (GenomeCov property), 94
FileFactory (class in sequana.snakertools), 153
filenames() (FileFactory property), 154
filter() (FastQ method), 108
filter_bool() (BAMSimul method), 141
filter_length() (BAMSimul method), 141
filter_length() (PacbioSubreads method), 134
filter_mapq() (PacbioMappedBAM method), 129
filter_vcf() (VCF_freebayes method), 165
Fixed_freebayes (class in sequana.freebayes_vcf_filter), 164
filters_params() (VCF_freebayes property), 165
FindAdaptersFromDesign (class in sequana.adapters), 84
findpos() (in module sequana.misc), 162
format_contigs_denovo() (FastA method), 115

G
G() (Coverage property), 102
gc_content() (Sequence method), 119
GC_skew() (DNA property), 116
gc_vs_coverage() (ChromosomeCoverageModule method), 169
gc_window_size() (GenomeCov property), 95
genbank_filename() (GenomeCov property), 95
GenomeCov (class in sequana.bedtools), 92
get_actg_content() (FastQC method), 111
get_adapter_by_identifier() (AdapterReader method), 83
get_adapter_by_index_name() (AdapterReader method), 83
get_adapter_by_index_seq() (AdapterReader method), 83
get_adapter_by_sequence() (AdapterReader method), 83
get_adapters_from_sample() (FindAdaptersFromDesign method), 85
get_args() (DoubleThresholds method), 100
get_centralness() (ChromosomeCov method), 98
get_complement() (Sequence method), 119
get_coverage() (PacbioMappedBAM method), 129
get_df() (MultiKrakenResults method), 126
get_file1() (FastQFactory method), 153
get_file2() (FastQFactory method), 153
get_flags() (SAMFlags method), 92
get_gaussians() (ChromosomeCov method), 98
get_gccorrelation() (ChromosomeCov method), 98
get_gis() (KrakenBuilder method), 129
get_kmer() (in module sequana.kmer), 119
get_lengths() (FastQ method), 109
get_lengths_as_dict() (FastA method), 115
get_max_gccorrelation() (ChromosomeCov method), 98
get_mean_contig_length() (Coverage method), 102
get_mean_nb_passes() (PacbioSubreads method), 134
get_mean_number_contigs() (Coverage method), 102
get_mean_reads_per_contig() (Coverage method), 102
get_meaning() (SAMFlags method), 92
get_number_of_ccs() (PacbioSubreads method), 134
get_occurences() (Sequence method), 119
get_percent_genome_sequenced() (Coverage method), 102
get_required_coverage() (Coverage method), 103
get_reverse() (Sequence method), 119
get_reverse_complement() (Sequence method), 119
get_rois() (ChromosomeCov method), 98
get_sample() (FindAdaptersFromDesign method), 85
get_sequana_adapters() (in module sequana.adapters), 85
get_size() (ChromosomeCov method), 98
get_stats() (ChromosomeCov method), 98
get_stats() (FastA method), 115
get_stats() (FastQC method), 111
get_stats() (GenomeCov method), 95
get_summary() (ChromosomeCov method), 98
get_summary() (Coverage method), 104
get_summary_string() (BUSCO method), 86
get_table() (Coverage method), 104
get_taxonomy_db() (KrakenResults method), 122
get_taxons_from_gis() (KrakenBuilder method), 129
get_variants() (VCF_freebayes method), 166
getlogdir() (PipelineManager method), 157
getlogdir() (PipelineManagerGeneric method), 158
getname() (PipelineManager method), 157
Sequana, Release 0.8.0

getName() (PipelineManagerGeneric method), 158
getrawdata() (PipelineManager method), 158
getrawdata() (PipelineManagerGeneric method), 158
getreportdir() (PipelineManager method), 158
getreportdir() (PipelineManagerGeneric method), 158
getwkdir() (PipelineManager method), 158
GZLineCounter (class in sequana.tools), 164

H
dhdr() (DoubleThresholds property), 101
header() (Repeats property), 118
high() (DoubleThresholds property), 101
high2() (DoubleThresholds property), 101
hist() (GenomeCov method), 95
hist_concordance() (PacbioMappedBAM method), 130
hist_GC() (BAMSimul method), 141
hist_GC() (PacbioMappedBAM method), 129
hist_GC() (PacbioSubreads method), 134
hist_length_repeats() (Repeats method), 118
hist_mean_polymerase_read_length() (Barcoding method), 145
hist_median_ccs() (PacbioMappedBAM method), 130
hist_nb_passes() (PacbioSubreads method), 134
hist_ORF_CDS_logscale() (DNA method), 116
hist_ORF_CDS_linearscale() (DNA method), 116
hist_polymerase_per_barcode() (Barcoding method), 145
hist_quality_perBarcode() (Barcoding method), 145
hist_read_length() (BAMSimul method), 142
hist_read_length() (PacbioMappedBAM method), 131
hist_read_length() (PacbioSubreads method), 136
hist_snr() (PacbioSubreads method), 137
histogram_GC_content() (FastQC method), 111
histogram_sequence_coordinates() (FastQC method), 111
histogram_sequence_lengths() (FastQC method), 111
html_id() (DataTable property), 172
html_id() (DataTableFunction property), 173

I
Identifier (class in sequana.fastq), 107
identifier() (Adapter property), 82
identifiers() (AdapterReader property), 83
imshow_qualities() (FastQC method), 111
include_avg_image() (SequanaBaseModule method), 167
index_names() (AdapterReader property), 83
index_sequence() (Adapter property), 82
index_sequences() (AdapterReader property), 84
init() (KrakenBuilder method), 129
init_roi_datatable() (CoverageModule method), 168
input_filename (GenomeCov attribute), 95
is_executable() (Module method), 156
is_fastq() (in module sequana.fastq), 114
is_joint() (VCF_freebayes property), 166
is_pipeline() (Module method), 156
ITOL (class in sequana.itol), 119

J
joining() (FastQ method), 109
JSON, 195

K
kraken_to_csv() (KrakenResults method), 122
kraken_to_json() (KrakenResults method), 122
kraken_to_krona() (KrakenResults method), 122
KrakenAnalysis (class in sequana.kraken), 124
KrakenBuilder (class in sequana.kraken_builder), 127
KrakenDownload (class in sequana.kraken), 125
KrakenHierarchical (class in sequana.kraken), 126
KrakenPipeline (class in sequana.kraken), 122
KrakenResults (class in sequana.kraken), 121
KronaMerger (class in sequana.krona), 126

L
L() (Coverage property), 102
launch_snpEff() (SnpEff method), 161
ldtr() (DoubleThresholds property), 101
length() (Repeats property), 118
lengths() (FastA property), 115
list_len_repeats() (Repeats property), 118
logo() (Module property), 156
longest_shustring() (Repeats property), 118
low() (DoubleThresholds property), 101
low2() (DoubleThresholds property), 101

M
md5() (Module method), 156
mean_quality() (Quality property), 146
message() (PipelineManager method), 158
Module, 195
Module (class in sequana.snaketools), 154
moving_average() (ChromosomeCov method), 98
moving_average() (module sequana.stats), 162
MultiKrakenResults (class in sequana.kraken), 126
multiqc_config() (Module property), 156

N
N() (Coverage property), 102
n_lines() (FastQ property), 109
n_reads() (FastQ property), 109
name() (Adapter property), 82
name() (Module property), 156
names() (FastA property), 115
names() (Repeats property), 118
next() (ChromosomeCov method), 98
next() (FastA method), 115
next() (FastQ method), 109
normalized_coverage() (ChromosomeCoverage-
Module method), 169
normpdf() (in module sequana.misic), 162

O
on_cluster() (in module sequana.misic), 162
ORF_pos() (DNA property), 116
overview() (Module property), 156

P
PacbioMappedBAM (class in sequana.pacbio), 129
PacbioSubreads (class in sequana.pacbio), 133
paired() (FastQFactory property), 153
paired() (PipelineManager property), 158
path() (Module property), 156
pathname() (FileFactory property), 154
pathnames() (FileFactory property), 154
pattern (Cigar attribute), 90
PBSim (class in sequana.pacbio), 140
pie_plot() (BUSCO method), 86
pipeline_names (in module sequana.snaketools), 160
PipelineManager (class in sequana.snaketools), 157
PipelineManagerGeneric (class in se-
quana.snaketools), 158
plot() (KrakenResults method), 122
plot() (Quality method), 148
plot() (SnakeMakeStats method), 160
plot_acgt_content() (FastQC method), 114
plot_all_skews() (DNA method), 116
plot_and_save() (SnakeMakeStats method), 160
plot_and_save_all() (Barcoding method), 145
plot_coverage() (ChromosomeCov method), 98
plot_GC_read_len() (BAMSimul method), 143
plot_GC_read_len() (PacbioMappedBAM
method), 131
plot_GC_read_len() (PacbioSubreads method), 138
plot_gc_vs_coverage() (ChromosomeCov
method), 99
plot_hist_coverage() (ChromosomeCov
method), 99
plot_hist_normalized_coverage() (Chromo-
someCov method), 99
plot_hist_zscore() (ChromosomeCov method), 99
plot_polymerase_per_barcode() (Barcoding
method), 145
plot_rois() (ChromosomeCov method), 99
plot_stacked_hist() (MultiKrakenResults
method), 126
plot_subreads_histogram() (Barcoding
method), 145
png_to_embedded_png() (SequanaBaseModule
method), 168
proba_to_quality_sanger() (in module se-
quana.phred), 148

Q
Quality (class in sequana.phred), 146
quality() (Quality property), 148
quality_to_proba_sanger() (in module se-
quana.phred), 148

R
random_selection() (PacbioSubreads method), 139
read() (ExpDesignBase method), 107
readme() (Module property), 156
realpaths() (FileFactory property), 154
record() (Variant property), 166
regions_of_interest() (ChromosomeCoverage-
Module method), 169
Repeats (class in sequana.sequence), 117
required_dir (SequanaBaseModule attribute), 168
requirements() (Module property), 156
reset() (BAMSimul method), 143
reset() (ChromosomeCov method), 100
reset() (PacbioMappedBAM method), 132
reset() (PacbioSubreads method), 140
rest2html() (in module sequana.misic), 162
resume() (Variant property), 166
reverse() (AdapterReader method), 84
reverse() (Sequence method), 119
reverse_and_save() (FastA method), 115
reverse_complement() (AdapterReader method), 84
reverse_complement() (Sequence method), 119
rewind() (FastQ method), 109
rewind() (VCF_freebayes method), 166
RNA (class in sequana.sequence), 116
rois() (ChromosomeCov property), 100
run() (ChromosomeCov method), 100
run() (KrakenAnalysis method), 125
run() (KrakenBuilder method), 129
run() (KrakenHierarchical method), 126
run() (KrakenPipeline method), 124
run() (PBSim method), 141
run() (RunningMedian method), 151
running_median() (ChromosomeCov method), 100
running_median() (in module sequana.running_median), 151
RunningMedian (class in sequana.running_median), 148

S

SAM, 195
SAM (class in sequana.bamtools), 92
SAMFlags (class in sequana.bamtools), 91
sample_names() (FindAdaptersFromDesign property), 85
samples() (Variant property), 166
save() (SequanaConfig method), 160
save_adapters_to_fasta() (FindAdaptersFromDesign method), 85
save_ctg_to_fasta() (FastA method), 115
save_summary() (PacbioSubreads method), 140
scatter_plot() (BUSCO method), 86
schema_config() (Module property), 157
score() (BUSCO property), 86
select_random_reads() (FastA method), 115
select_random_reads() (FastQ method), 109
sequana.adapters (module), 81
sequana.assembly (module), 86
sequana.bamtools (module), 90
sequana.bedtools (module), 92
sequana.cigar (module), 86
sequana.coverage (module), 101
sequana.databases (module), 104
sequana.datatools (module), 170
sequana.expdesign (module), 105
sequana.fasta (module), 115
sequana.fastq (module), 107
sequana.freebayes_vcf_filter (module), 164
sequana.iotools (module), 120
sequana.itol (module), 119
sequana.kmer (module), 119
sequana.kraken (module), 121
sequana.kraken_builder (module), 127
sequana.krona (module), 126
sequana.misc (module), 162
sequana.modules_report.bamqc (module), 168
sequana.modules_report.base_module (module), 167
sequana.modules_report.coverage (module), 168
sequana.pacbio (module), 129
sequana.phred (module), 145
sequana.resources.data (module), 170
sequana.running_median (module), 148
sequana.sequence (module), 116
sequana.snakertools (module), 151
sequana.snpeff (module), 161
sequana.stats (module), 162
sequana.tools (module), 163
sequana.utils.datatables_js (module), 171
sequana_data() (in module sequana.datatools), 170
SequanaBaseModule (class in sequana.modules_report.base_module), 167
SequanaConfig (class in sequana.snakertools), 160
Sequence (class in sequana.sequence), 118
sequence() (Adapter property), 82
sequence() (Sequence property), 119
sequences() (AdapterReader property), 84
sequences() (FASTA property), 115
set_links_to_column() (DataTableFunction method), 173
set_tooltips_to_column() (DataTableFunction method), 173
show() (KrakenPipeline method), 124
Snakefile, 195
snakefile() (Module property), 157
SnakeMakeStats (class in sequana.snakertools), 158
SnpEff (class in sequana.snpeff), 161
split_chunks() (FastQ method), 109
split_lines() (FastQ method), 109
stats() (Cigar method), 90
stats() (FastQ method), 109
stats() (PacbioSubreads property), 140
stats() (Sequence method), 119
StatsBAM2Mapped (class in sequana.tools), 163
STD() (ChromosomeCov property), 97
strand_ratio() (in module sequana.freebayes_vcf_filter), 166
stride() (PacbioSubreads method), 140
subcoverage() (ChromosomeCoverageModule method), 169
summary() (BUSCO method), 86
summary() (PacbioSubreads method), 140
switch_header_to_gi() (ENADownload method), 105

target_distribution() (PBsim method), 141
taxons() (KrakenResults property), 122
textwrap() (in module sequana.misc), 162
threshold() (DNA property), 116
threshold() (Repeats property), 118
thresholds (ChromosomeCov attribute), 100
to_csv() (ChromosomeCov method), 100
to_csv() (Filtered_freebayes method), 164

to_csv() (GenomeCov method), 95

to_dict() (AdapterReader method), 84

to_fasta() (AdapterReader method), 84

to_fasta() (BAMSimul method), 145

to_fasta() (FastA method), 115

to_fasta() (FastQ method), 109

to_fasta() (PacbioMappedBAM method), 132

to_fasta() (PacbioSubreads method), 140

to_fastq() (BAMSimul method), 145

to_fastq() (PacbioMappedBAM method), 133

to_fastq() (PacbioSubreads method), 140

to_html() (StatsBAM2Mapped method), 163

to_igv_chrom_size() (FastA method), 115

to_js() (KrakenResults method), 122

to_kmer_content() (FastQ method), 109

to_krona() (FastQ method), 109

to_tsv() (KronaMerger method), 127

to_vcf() (Filtered_freebayes method), 164

type_filter() (DNA property), 116

type_window() (DNA property), 116

types (Cigar attribute), 90

U

upload() (ITOL method), 120

V

Variant (class in sequana.freebayes_vcf_filter), 166

variants() (Filtered_freebayes property), 165

VCF, 195

vcf() (Filtered_freebayes property), 165

VCF_freebayes (class in sequana.freebayes_vcf_filter), 165

version() (Module property), 157

W

wget() (in module sequana.misc), 162

window() (DNA property), 116

window_size() (GenomeCov property), 95

Y

YAML, 195

YamlDocParser (class in sequana.iotools), 120

Z

zscore_distribution() (ChromosomeCoverage-Module method), 169