ngstoolkit Documentation

Release 0.19.3

Andre Rendeiro

Oct 18, 2019

CONTENTS

`ngs_toolkit` is a Python library for the analysis of NGS data.

Its goals are to provide a highly customizable set of objects and tools that interact with each other to create data processing and analysis workflows in both a interactive and scripted way.

`ngs-toolkit` is unique in the following aspects:

- Includes tried-and-tested (and published) workflows for end-to-end analysis of NGS data, while at the same time allowing high customization;
- Tailored for well-established NGS data types, but supporting arbitrary data types;
- Its target audience are mid-level computational biologists who want to "get it done" and focus on interpretation of results. At the same time, it allows running workflows with minimal programming experience.

`ngs-toolkit` is reaching maturity, with a stable API (from version 0.14.0 on), improving documentation and increasing test coverage.

Head to the *Installation* to see installation instructions, to *Usage* for quick use, or have a look at the catalogue of available functions in the *API*.

CHAPTER

ONE

CONTENTS

1.1 Installation

1.1.1 With pip

`ngs_toolkit` is available for Python 3 only. Is is tested in Python 3.6 and 3.7.

To install, simply do:

```
pip install ngs-toolkit
```

you might need to add a `--user` flag if not root or running in a virtual environment.

This will install all the Python dependencies needed too. See here a list of all Python dependencies used.

To install the latest development version:

```
pip install git+https://github.com/afrendeiro/toolkit.git#egg=ngs-toolkit
```

1.1.2 Using a conda environment

Get the latest Python 3 installation of miniconda from the conda website and follow the instructions for installation and activation of the environment.

Setup the bioconda channel:

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

Install non-Python dependencies:

```
conda install -y bedtools==2.27.1
conda install -y ucsc-twobittofa
conda install -y bioconductor-deseq2
conda install -y bioconductor-cqn
```

And then install the `ngs-toolkit` library with pip (available only through PyPi).

```
pip install ngs-toolkit
```

1.1.3 Non-Python requirements

`ngs_toolkit` makes use of some non-Python dependencies.

- bedtools: version should be at least 2.27.1

The following are highly recommended only for some data or analysis types:

- R and some bioconductor libraries (optional): - DESeq2 (optional): used for differential testing of genes/regulatory elements and variance stabilization transformation of data. - cqn (optional): used for GC-content aware normalization of NGS data.
- Kent tools (optional): - the `twoBitToFa` binary from UCSC's Kent bioinformatics toolkit is used to convert between the 2bit and FASTA formats.

For region-based enrichment analysis, you may also want to have the following software installed (entirely optional):

- MEME suite
- HOMER motif analysis
- LOLA R package

You can see how to install all requirements in an Ubuntu-based system in the provided Dockerfile.

1.1.4 Docker

A Docker image containing `ngs_toolkit` and its dependencies is also available: https://hub.docker.com/r/afrendeiro/ngs-toolkit

To pull the image and run a module for example in this way:

```
docker pull afrendeiro/ngs-toolkit
docker run ngs-toolkit python3 -m ngs_toolkit.recipes.ngs_analysis --help
```

You can also run an interactive session of `ngs_toolkit` based on the docker image on Gitpod.

The Dockerfile that produced the image is available in the github repository: https://github.com/afrendeiro/toolkit/blob/master/Dockerfile

1.2 Quick usage

1.2.1 Interactive usage through the API

To use a particular class or function from the toolkit, simply import it following the structure of the library:

```
from ngs_toolkit import ATACSeqAnalysis
from ngs_toolkit.utils import log_p_values
```

The *ngs_toolkit.analysis.Analysis* and their data type-specific children are the main drivers of the workflow, storing attributes and providing various methods through an OOP interface:

```
from ngs_toolkit.demo import generate_project

an = generate_project(data_type="ATAC-seq", sample_input_files=True)
an.measure_coverage()
an.normalize()
```

(continues on next page)

(continued from previous page)

```
an.unsupervised_analysis()
an.differential_analysis()
an.plot_differential()
an.get_peak_gene_annotation()
an.annotate_features()
an.differential_enrichment(steps=['enrichr'])
an.plot_differential_enrichment()
```

1.2.2 Running recipes through the command-line interface

`ngs_toolkit` also has some command-line programs on some commonly used workflows (here called `recipes`), which can be run in the following manner:

```
PEP=`python -m ngs_toolkit.recipes.generate_project --sample-input-files True`
python -m ngs_toolkit.recipes.ngs_analysis $PEP
```

This example is roughly equivalent to the on above with interactive usage.

1.3 Examples

1.3.1 Analysis example

The following is an example of how to use `ngs_toolkit` in a ATAC-seq project. While straightforward, it still allows considerable customization due to the modularity of the toolkit and the parametrization of most functions (this example uses default values everywhere nonetheless).

We have the following PEP project config YAML file:

```
project_name: example_project
project_description: example_project
username: user
email: user@cemm.oeaw.ac.at
metadata:
  output_dir: /scratch/lab_bock/shared/projects/example_project
  results_subdir: data
  submission_subdir: submission
  pipeline_interfaces: /home/user/workspace/open_pipelines/pipeline_interface.yaml
  sample_annotation: /scratch/lab_bock/shared/projects/example_project/metadata/
↪annotation.csv
  sample_subannotation: /scratch/lab_bock/shared/projects/example_project/metadata/
↪sample_subannotation.csv
  comparison_table: /scratch/lab_bock/shared/projects/example_project/metadata/
↪comparison_table.csv
sample_attributes:
  - sample_name
  - genotype
  - replicate
group_attributes:
  - genotype
  - replicate
data_sources:
  bsf: /path/to/samples/{flowcell}/{flowcell}_{lane}#{sample_name}.bam
```

(continues on next page)

(continued from previous page)

```
genomes:
  human: hg19
trackhubs:
  trackhub_dir: /path/to/public_html/user/example_project/
  url: http://root-url.com/example_project
```

The following sample annotation CSV file:

Table 1: Annotation table for example

sample_name	genotype	replicate	organism	flowcell	lane
ATAC-seq_KOA_r1	KO_A	1	human	C0RQ31ACXXX	1
ATAC-seq_KOA_r2	KO_A	2	human	C0RQ31ACXXX	1
ATAC-seq_KOB_r1	KO_B	1	human	C0RQ31ACXXX	1
ATAC-seq_KOB_r2	KO_B	2	human	C0RQ31ACXXX	1
ATAC-seq_WT_r1	WT	1	human	C0RQ31ACXXX	1
ATAC-seq_WT_r2	WT	2	human	C0RQ31ACXXX	1

And the following comparison table:

Table 2: Comparison table for example

comparison_name	comparison_side	sample_name	sample_group
KOA_vs_WT	1	ATAC-seq_KOA_r1	KO_A
KOA_vs_WT	1	ATAC-seq_KOA_r2	KO_A
KOA_vs_WT	0	ATAC-seq_WT_r1	WT
KOA_vs_WT	0	ATAC-seq_WT_r2	WT
KOB_vs_WT	1	ATAC-seq_KOB_r1	KO_B
KOB_vs_WT	1	ATAC-seq_KOB_r2	KO_B
KOB_vs_WT	0	ATAC-seq_WT_r1	WT
KOB_vs_WT	0	ATAC-seq_WT_r2	WT

ATAC-seq analysis example

```
import os
from ngs_toolkit.atacseq import ATACSeqAnalysis

# Start project and analysis objects
analysis = ATACSeqAnalysis(
    from_pep=os.path.join("metadata", "project_config.yaml"))

# Generate consensus peak set and annotate it
## get consensus peak set from all samples
analysis.get_consensus_sites()
## annotate peak set with genomic context
analysis.get_peak_genomic_location()
## annotate peak set with chromatin context
analysis.get_peak_chromatin_state(
    os.path.join(
        analysis.data_dir,
        "external",
        "E032_15_coreMarks_mnemonics.bed"))
## annotate peak set with genes
```

(continues on next page)

(continued from previous page)

```
analysis.get_peak_gene_annotation()

# Use accessibility quantitatively
## get coverage values for each peak in each sample of ATAC-seq
analysis.measure_coverage()

# Normalize accessibility (quantile normalization + GC correction, requires cqn R␣
↪library)
analysis.normalize(method="cqn")

# Annotate normalized accessibility with sample and region info
# # annotate dataframe with peak metadata
analysis.annotate_features()
# # annotate dataframe with sample metadata
analysis.accessibility = analysis.annotate_samples()

# Save analysis object
analysis.to_pickle()


# UNSUPERVISED ANALYSIS
# # plot pairwise sample correlations,
# # perform dimensionality reduction (MDS, PCA)
# # and plot samples in this spaces, annotated with their attributes
analysis.unsupervised_analysis()


# SUPERVISED ANALYSIS
# # differential analysis with DESeq2
analysis.differential_analysis()

# # Save analysis object
analysis.to_pickle()

# # plot scatter, volcano, MA, heatmaps on the differential regions
# # by groups and with individual samples, with normalized values
# # and scalled values (Z-score).
analysis.plot_differential(
    alpha=0.05,
    corrected_p_value=True,
    fold_change=1)

# # perform enrichment analysis on differnetial region sets
# # using LOLA, MEME-AME, HOMER and Enrichr
analysis.differential_enrichment(
    directional=True,
    max_diff=1000,
    sort_var="pvalue")

# # for each type of enrichment results,
# # plot bar and scatter plots of odds ratio vs p-value,
# # heatmaps of enrichment across terms for each comparison
# # and comparison correlation in enrichment terms
analysis.plot_differential_enrichment()
```

1.4 Concepts

A few notes on the way some of the library and its objects were designed to be used.

1.4.1 Analysis objects

The `Analysis` object and its data-type specific dependents are central to the usage of `ngs-toolkit`. These objects hold attributes and functions relevant to the analysis, such as `Sample` objects (and their attributes), Dataframes with numerical values, and others.

Leveraging on the PEP format

One easy and recommended way to instantiate `Analysis` objects is with a `PEP Project` file. This has several advantages:

- Usage of the language-agnostic PEP format to store a project description and interoperability with other tools (see https://github.com/pepkit for other tools);
- Initialization of project-specific variables into the `Analysis` object that are derived from the PEP. Examples: analysis samples, genome(s), sample and sample group attributes, sample comparison table.

The example below shows how this works:

```
>>> from ngs_toolkit import Analysis
>>> an = Analysis(from_pep="my_project/metadata/project_config.yaml")
[INFO] > Setting project's 'sample_attributes' as the analysis 'sample_attributes'.
[INFO] > Setting project's 'group_attributes' as the analysis 'group_attributes'.
[INFO] > Setting project's 'comparison_table' as the analysis 'comparison_table'.
[INFO] > Setting analysis organism as 'mouse'.
[INFO] > Setting analysis genome as 'mm10'.
>>> print(an)
Analysis 'my_project' with 12 samples of organism 'mouse' (mm10).
```

Note: The verbosity of `ngs-toolkit` can be controlled

See the section on logging to control the verbosity of `ngs-toolkit`.

Reasonable defaults with full customization

Functions in the `Analysis` object are aware of these attributes and will use them by default, making calling the functions very simple (other overiding arguments can be passed though).

In the example below, we will generate a consensus peak set for ATAC-seq analyss using the `get_consensus_sites` function. This will demonstrate several things that "come for free":

```
>>> from ngs_toolkit import ATACSeqAnalysis
>>> an = ATACSeqAnalysis(from_pep="my_project/metadata/project_config.yaml")
[INFO] > Setting project's 'sample_attributes' as the analysis 'sample_attributes'.
[INFO] > Setting project's 'group_attributes' as the analysis 'group_attributes'.
[INFO] > Setting project's 'comparison_table' as the analysis 'comparison_table'.
[INFO] > Subsetting samples for samples of type 'ATAC-seq'.
[INFO] > Subsetting comparison_table for comparisons of type 'ATAC-seq'.
```

(continues on next page)

(continued from previous page)

```
[INFO] > Setting analysis organism as 'mouse'.
[INFO] > Setting analysis genome as 'mm10'.
>>> an.get_consensus_sites()
```

- even though the PEP project includes samples from several data types (ATAC-, ChIP- and RNA-seq), the current analysis will only consider ATAC-seq samples.
- the necessary files with peak calls for each sample are not specified - `ngs-toolkit` knows where to find them;
- a BED file with ENCODE blacklisted regions will not be given, but these regions will be filtered out - `ngs-toolkit` will download this and use it. No static files are distributed with the package.
- related to the above, the correct blacklist file is downloaded because the genome assembly for the project is infered from the samples - even though it is not directly specified.

Workflow

Most functions of the `Analysis` object will take some input (usually a dataframe), apply some transformation and assign the result to a variable of the same Analysis object.

To see what variable has been assigned within a given function check the relevant function in the API, specifically the *Variables* value. Some functions will assign attributes that are used almost ubiquitily. See the common attributes section for some examples.

High-level functions will also often assign their outputs to the object itself. To see which attribute holds it, note the `Attributes` section of the respective function documentation. Assignment allows the exchange of information between analysis steps without the user always providing all required inputs, which would make using such a toolkit quite verbose.

The example below illustrates this:

```
>>> from ngs_toolkit import ATACSeqAnalysis
>>> an = ATACSeqAnalysis(from_pep="my_project/metadata/project_config.yaml")
>>> print(an)
'ATAC-seq' analysis 'test-project_ATAC-seq_mm10_1_100_1' with 2 samples of organism
↪'mouse' (mm10).
>>> an.get_consensus_sites()
>>> an.measure_coverage()
>>> print(an.matrix_raw.head())
                        S1_a1  S2_a2
region
chr1:42447241-42447627    955   2211
chr1:44445678-44446750   1939   2122
chr1:44743959-44744926   1264   1443
chr1:90513210-90513978   1262   1354
chr1:93565764-93567191    911    892
>>> an.normalize()
>>> print(an.matrix_norm.head())
region                       S1_a1      S2_a2
chr1:42447241-42447627  12.681954  13.822151
chr1:44445678-44446750  13.703582  13.762881
chr1:44743959-44744926  13.086324  13.206576
chr1:90513210-90513978  13.084040  13.114743
chr1:93565764-93567191  12.613915  12.512715
```

All three `get_consensus_sites`, `measure_coverage` and `normalize` build on the output of each other, but the user doesn't have to specify the input to any. Changing either the name of the attribute that stores either output

or the location of files outputed is nonetheless easy.

Many functions also have a `save` argument which will save the result as a `CSV` file.

Common attributes

To allow a uniform usage across different data types and analysis types, a few but important attributes of the `Analysis` object and its derivatives have naming conventions:

- `data_type`: The type of data of the analysis. Matches the object type.
- `matrix_raw`: A dataframe of raw, unnormalized values of shape (features, samples)
- `matrix_norm`: A dataframe of normalized values of shape (features, samples)
- `quantity`: The name of the units of the values measured. E.g. "expression" for RNA-seq or "accessibility" for ATAC-seq
- `var_unit_name`: The name of the variables measured. E.g. "gene" for RNA-seq or "region" for ATAC-seq or ChIP-seq
- `norm_method`: The method used to normalize the `matrix_norm` dataframe
- `thresholds`: A dictionary with keys "log_fold_change" and "p_value" storing thresholds used in the analysis

1.4.2 Comparison table

`ngs-toolkit` has functions to perform supervised differntial comparisons between groups of samples. The sample groupings are specified in a CSV file called `comparison_table`.

An example of a typical "case vs control" comparison table is given below:

Table 3: Typical example of comparison_table

comparison_name	comparison_side	sample_name	sample_group
KOA_vs_WT	1	ATAC-seq_KOA_r1	KO_A
KOA_vs_WT	1	ATAC-seq_KOA_r2	KO_A
KOA_vs_WT	0	ATAC-seq_WT_r1	WT
KOA_vs_WT	0	ATAC-seq_WT_r2	WT
KOB_vs_WT	1	ATAC-seq_KOB_r1	KO_B
KOB_vs_WT	1	ATAC-seq_KOB_r2	KO_B
KOB_vs_WT	0	ATAC-seq_WT_r1	WT
KOB_vs_WT	0	ATAC-seq_WT_r2	WT

Each row is reserved for a given sample. Samples of the same group (typically replicates) should have the same value of "sample_group" and same "comparison_side". The group of interest (comparison foreground) should have a value of 1 as "comparison_side" and the background a value of 0. Finally, the comparison will be labeled with the value of "comparison_name", which should be constant for all samples in both foreground and background groups.

For an all-vs-all group comparison, I recommend labeling all background sample groups as a new group in the following manner:

Table 4: "All-vs-all" example of comparison table

comparison_name	comparison_side	sample_name	sample_group
celltypeA	1	ATAC-seq_celltypeA_r1	ct_A
celltypeA	1	ATAC-seq_celltypeA_r2	ct_A
celltypeA	0	ATAC-seq_celltypeB_r1	ct_A_background
celltypeA	0	ATAC-seq_celltypeB_r2	ct_A_background
celltypeA	0	ATAC-seq_celltypeC_r1	ct_A_background
celltypeA	0	ATAC-seq_celltypeC_r2	ct_A_background
celltypeB	1	ATAC-seq_celltypeB_r1	ct_B
celltypeB	1	ATAC-seq_celltypeB_r2	ct_B
celltypeB	0	ATAC-seq_celltypeA_r1	ct_B_background
celltypeB	0	ATAC-seq_celltypeA_r2	ct_B_background
celltypeB	0	ATAC-seq_celltypeC_r1	ct_B_background
celltypeB	0	ATAC-seq_celltypeC_r2	ct_B_background

Additional useful columns are *data_type* (to subset comparisons based on type of NGS data), *comparison_type* to specify the type of comparison to perform (e.g. one of 'differential' or 'peaks') and *toggle* for subsetting comparisons to perform.

Note: Hyphens and other symbols in comparison_table

Since differential comparisons are perfomed using DESeq2, R is used (throught the Python-R interface library rpy2). ngs_toolkit will create the required tables by DESeq2 which includes names of samples and comparisons as dataframe columns. Unfortunately due to the way R handles column names, these get changed.

In the future this will be accounted for but for now avoid using hyphens and any other symbols as values for sample names or groups.

1.4.3 Low-level functions - `utils`

Functions from Analysis objects are generally pretty high level functions, often performing several tasks by calling other more general-purpose functions. However, one of the concepts I really wanted to have is that the user retains as much control as they wish.

They may choose to use the high level functions which generally provide sensible defaults, or retain more control and build their analysis pipeline from the lower level helper functions.

One example: calling `ATACSeqAnalysis.normalize()` will by default run 3-4 other functions to return a quantile normalized, GC-corrected, log-transformed output - a fairly complex normalization procedure but made simple by providing sensible defaults.

A user may easily change the procedure by choosing one of the ~4 types of normalization using keyword arguments or implement an alternative method which can be plugged in to the next step of the analysis.

In the future the low level functions will be moved to *ngs_toolkit.utils* and the data type-specific modules will have only classes and functions specific to those data which are usually more high-level.

1.5 Configuration, logging and versioning

1.5.1 Configuration

`ngs_toolkit` uses a YAML configuration file.

While entirely optional, this allows the user to specify preferences, patterns and allows usage across different computing environments.

The user can provide its own configuration in two ways:

- In a YAML file located in `$HOME/.ngs_toolkit.config.yaml`;
- A user provided file given during interactive runtime passed to `ngs_toolkit.setup_config()`.

If more than one is given values in the configuration files will be updated in the following order:

1. A minimal configuration file from the package data;
2. The user provided file in `$HOME/.ngs_toolkit.config.yaml`;
3. The user provided file passed to `ngs_toolkit.setup_config()`.

To see how to structure the YAML file, see section below.

Example configuration files

To see all available configuration fields have a look at the default configuration file: https://github.com/afrendeiro/toolkit/blob/master/ngs_toolkit/config/default.yaml#L1

For a full example of a fully configured file have a look at the example configuration file: https://github.com/afrendeiro/toolkit/blob/master/ngs_toolkit/config/example.yaml#L1

However, the configuration file does not need to include all fields. Below is a minimal example of a configuration file.

```
username: user
email: user@mail.com
website_root: userwebsite.web.com
preferences:
  # For the next item, environment variables are formatted if they are of the form $
↪{VAR}
  root_reference_dir: ${USER}/reference_data
  root_projects_dir: ${USER}/projects
  default_genome_assemblies:
    - human: hg38
    - mouse: mm10
  # Below is the name of the divvy package configuration (http://divvy.databio.org/en/
↪latest/)
  computing_configuration: 'slurm'
sample_input_files:
  ATAC-seq:
    aligned_filtered_bam: "{data_dir}/{sample_name}/mapped/{sample_name}.bowtie2.
↪filtered.bam"
    peaks: "{data_dir}/{sample_name}/peaks/{sample_name}_peaks.narrowPeak"
    summits: "{data_dir}/{sample_name}/peaks/{sample_name}_summits.narrowPeak"
  ChIP-seq:
    aligned_filtered_bam: "{data_dir}/{sample_name}/mapped/{sample_name}.bowtie2.
↪filtered.bam"
  CNV:
```

(continues on next page)

(continued from previous page)

```
    log2_read_counts: "{data_dir}/{sample_name}/{sample_name}_{resolution}/
↪CNAprofiles/log2_read_counts.igv"
  RNA-seq:
    aligned_filtered_bam: "{data_dir}/{sample_name}/mapped/{sample_name}.bowtie2.
↪filtered.bam"
    bitseq_counts: "{data_dir}/{sample_name}/quantification/{sample_name}_bitseq.tsv"
```

Note: *Not all elements are required*

In fact none of it is required, but it is recommended to have a look at the template configuration file and set custom options.

1.5.2 Logging

`ngs_toolkit` will log its operations and errors using the Python standard logging library.

This will happen by default to standard output (sys.stdout) but also to a file in `$HOME/.ngs_toolkit.log.txt`.

The location of the log file and the level of events to be reported can be customized in the `ngs_toolkit.setup_logger()` function.

1.5.3 Versioning

`ngs_toolkit` will by default timestamp every output it produces (CSV and figure files).

This behaviour can be controlled independently for tables and figures by setting the respective values of the configuration file:

```
preferences:
  report:
    timestamp_figures: False
    timestamp_tables: False
```

1.6 Analysis reports

Each analysis object in the `ngs_toolkit` will by default record the outputs it produces (e.g. tables, figures). This allows the collection of all outputs in a standardized way and the generation of an HTML report.

By default the location of the report will be in: `<root_directory>/<analysis_name>.analysis_report.html`

Every time a new output is produced, a new report is generated, in a way that analysis progress can be easily monitored in a user-friendly way by simply refreshing the HTML report file. This continuous generation behaviour can be controlled in the configuration file.

The recording behaviour can also be controlled independently for tables and figures by setting the respective values of the configuration file:

```
preferences:
  report:
    record_figures: True
```

(continues on next page)

(continued from previous page)

```
    record_csv: True
    continuous_generation: True
```

The report will by default be generated in the root of the project directory, but this can be controlled by manually calling the *ngs_toolkit.analysis.Analysis.generate_report()* function at the user's will.

1.7 Distributed computing

1.7.1 divvy

Certain functions in the `ngs_toolkit` toolkit can make use of distributed computing. To achieve this for a variety of computing configurations it uses the divvy library.

Divvy provides an abstract way of submitting a job to various job managers by shipping job templates for each configuration.

When `divvy` starts, a configuration is chosen (the `compute_configuration` attribute) and that template gets filled with the attributes of the job - the code to be executed, the resouce requirements and others (e.g. "cores", "mem", "time" attributes).

To see all supported compute configurations run:

```
divvy list
```

For more information on how to configure `divvy`, see its documentation: http://divvy.databio.org/

To let `ngs_toolkit` know which `divvy` configuration to use by default, modify the following section in the `ngs_toolkit` configuration file:

```
preferences:
  # The next item is the default computing configuration to use from divvy.
  # Run "divvy list" to see all options.
  # See more here: http://code.databio.org/divvy/
  computing_configuration: 'slurm'
```

This will make `ngs_toolkit` send jobs to a slurm cluster if wanted.

All functions that allow running a task in a distributed manner have a `distributed` keyword argument.

In addition, they also accept additional keyword arguments (*kwargs* in the function signature) where additional options can be passed. These options must match fields available to format of the currently selected `compute_configuration`.

1.7.2 Sending jobs and collecting output

Performing a taks in a distributed manner can therefore be as simple as calling the desired function with `distributed=True`. Jobs will be sent to the job manager of the chosen computing configuration.

However, since the jobs are often run individually for a sample/group of samples, functions called with `distributed=True` may not return the same output as `distributed=False`.

For that reason, for all such functions, there is a reciprocal function of identical name as the first prefixed with `collect`.

```
from ngs_toolkit.demo import generate_project
an = generate_project(sample_input_files=True)
an.measure_coverage(distributed=True)
coverage = collect_coverage()
```

Implementing automatic collection of job outputs in part of future plans.

1.7.3 Example

The *ngs_toolkit.atacseq.ATACSeqAnalysis.measure_coverage()* function has `distributed` and `kwargs` options.

This provides code portability and allows customization of various aspects of the jobs:

```
from ngs_toolkit.demo import generate_project
an = generate_project(sample_input_files=True)
# in serial
cov1 = an.measure_coverage()
# as slurm jobs (because the config computing_configuration is set to 'slurm')
an.measure_coverage(distributed=True)
cov2 = collect_coverage()
# confirm the output is the same
assert (cov2 == cov1).all().all()
```

```
# as slurm jobs to a particular queue and more memory
an.measure_coverage(distributed=True, partition="longq", mem=24000)
# here 'partition' and 'mem' are attributes of the slurm divvy template
# and not magic attributes
```

1.8 Manager programs

***ngs_toolkit* comes with two programs that provide a command line interface (CLI):**

- `projectmanager` handles the creation and execution of a *looper* project, providing sensible configuration templates and git-enabled tracking of changes.
- `trackmanager` handles the creation of a UCSC trackhub or IGV link for ATAC/ChIP-seq data based on bigWig files created by `looper` pipelines.

Here you can see the command-line usage instructions for the main looper command and for each subcommand:

1.8.1 projectmanager

```
usage: projectmanager [-h] {create,recipe} ...

projectmanager - A project manager.

positional arguments:
  {create,recipe}
    create         Create project.
    recipe         Run ngs_toolkit recipe for a given project.
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help       show this help message and exit

https://github.com/afrendeiro/toolkit
```

projectmanager::create

```
usage: projectmanager create [-h] [-r ROOT_DIR] [-d] [--overwrite]
                             project_name

Create project.

positional arguments:
  project_name          Project name.

optional arguments:
  -h, --help            show this help message and exit
  -r ROOT_DIR, --root-dir ROOT_DIR
                        Root directory to create projects.
  -d, --dry-run         Don't actually do anything.
  --overwrite           Don't overwrite any existing directory or file.
```

projectmanager::recipe

```
usage: projectmanager recipe [-h] recipe_name project_config

Run recipe.

positional arguments:
  recipe_name     Recipe name.
  project_config  Project config.

optional arguments:
  -h, --help      show this help message and exit
```

1.8.2 trackmanager

```
usage: trackmanager [-h] [-a [ATTRIBUTES]] [-c COLOR_ATTRIBUTE] [-r] [-l]
                    project_config_file

positional arguments:
  project_config_file

optional arguments:
  -h, --help            show this help message and exit
  -a [ATTRIBUTES], --attrs [ATTRIBUTES]
                        Sample attributes (annotation sheet columns) to use to
                        order tracks. Add attributes comma-separated with no
                        whitespace.
  -c COLOR_ATTRIBUTE, --color-attr COLOR_ATTRIBUTE
                        Sample attribute to use to color tracks with. Default
```

(continues on next page)

(continued from previous page)

```
                        is first attribute passed.
  -r, --overlay-replicates
                        Whether replicate samples should be overlaied in same
                        track. Default=False.
  -l, --link            Whether bigWig files should be soft-linked to the
                        track database directory. Default=False.
```

Note: *Copying vs linking bigWig files files in trackmanager*

The intention of trackmanager is to create a hierarchy of files in a HTTP server which can be used by genome browsers. This requires files (and their parent directories) to be readable and executable. When soft-linking files, they will retain the permission attributes of the original files and this may not be appropriate to serve through a server. Be aware that copying or linking these files does not always works (manual movement of files might be required).

Note: *Changing permissions of files and directories in bigwig directory*

Trackmanager will try to change the permissions of the bigwig files and their parent directories to allow reading and execution by everyone. Be aware that this does not always works (manual permission changes might be required).

1.9 Recipes

`ngs_toolkit` provides scripts to perform routine tasks on NGS data - they are called `recipes`.

Recipes are distributed with `ngs_toolkit` and can be seen in the github repository.

To make it convenient to run the scripts on data from a project, recipes can also be run with the command `projectmanager recipe <recipe_name> <project_config.yaml>`.

1.9.1 ngs_toolkit.recipes.ngs_analysis

Perform full end-to-end analysis of ATAC-seq, ChIP-seq or RNA-seq data.

Produces quantification matrices, normalizes them, performes unsupervised and supervised analysis as well as enrichment analyisis of differential features, all accompaigned with powerful visualizations.

Supervised analysis will only be performed if PEP configuration file contains a comparison table field.

In addition, this recipe uses variables provided in the project configuration file `project_name`, `sample_attributes` and `group_attributes`.

```
usage: python -m ngs_toolkit.recipes.ngs_analysis [-h] [-n NAME]
                                                  [-o RESULTS_DIR]
                                                  [-t {ATAC-seq,RNA-seq,ChIP-seq}]
                                                  [-q] [-a ALPHA]
                                                  [-f ABS_FOLD_CHANGE]
                                                  config_file
```

Positional Arguments

config_file YAML project configuration file.

Named Arguments

-n, --analysis-name Name of analysis. Will be the prefix of output_files. By default it will be the name of the Project given in the YAML configuration.

-o, --results-output Directory for analysis output files. Default is 'results' under the project roort directory.

Default: "results"

-t, --data-type Possible choices: ATAC-seq, RNA-seq, ChIP-seq

Data type to restrict analysis to. Default is to run separate analysis for each data type.

-q, --pass-qc Whether only samples with a 'pass_qc' value of '1' in the annotation sheet should be used.

Default: False

-a, --alpha Alpha value of confidence for supervised analysis.

Default: 0.05

-f, --fold-change Absolute log2 fold change value for supervised analysis.

Default: 0

1.9.2 ngs_toolkit.recipes.call_peaks

Call peaks for ChIP-seq samples given a comparison table mapping foreground-background relationships between samples.

```
usage: python -m ngs_toolkit.recipes.call_peaks [-h] [-c COMPARISON_TABLE]
                                                [-t] [-qc] [-j]
                                                [-o RESULTS_DIR]
                                                config_file
```

Positional Arguments

config_file YAML project configuration file.

Named Arguments

-c, --comparison-table Comparison table to use for peak calling. If not provided will use a filenamed *comparison_table.csv* in the same directory of the given YAML Project configuration file.

-t, --only-toggle Whether only comparisons with 'toggle' value of '1' or 'True' should be performed.

Default: False

-qc, --pass-qc Whether only samples with a 'pass_qc' attribute should be included. Default is `False`.

Default: False

-j, --as-jobs Whether jobs should be created for each sample, or it should run in serial mode.

Default: False

-o, --results-output Directory for analysis output files. Default is 'results' under the project root directory.

Default: "results"

1.9.3 ngs_toolkit.recipes.coverage

A helper script to calculate the read coverage of a BAM file in regions from a BED file. Ensures the same order and number of lines as input BED file.

```
usage: python -m ngs_toolkit.recipes.coverage [-h] [--no-overwrite]
                                              bed_file bam_file output_bed
```

Positional Arguments

bed_file Input BED file with regions to quantify.

bam_file Input BAM file with reads.

output_bed Output BED file with counts for each region.

Named Arguments

--no-overwrite Whether results should not be overwritten if existing.

Default: True

1.9.4 ngs_toolkit.recipes.deseq2

Perform differential expression using DESeq2 by comparing sample groups using a formula.

```
usage: python -m ngs_toolkit.recipes.deseq2 [-h]
                                            [--output_prefix OUTPUT_PREFIX]
                                            [--formula FORMULA]
                                            [--alpha ALPHA] [-d] [--overwrite]
                                            [--no-save-inputs]
                                            work_dir
```

Positional Arguments

work_dir Working directory. Should contain required files for DESeq2.

Named Arguments

--output_prefix Prefix for output files.

Default: "differential_analysis"

--formula	R-style formula for differential expression. Default = '~ sample_group'. Default: "~ sample_group"
--alpha	Significance level to call differential expression. All results will be output anyway. Default: 0.05
-d, --dry-run	Don't actually do anything. Default: False
--overwrite	Don't overwrite any existing directory or file. Default: False
--no-save-inputs	Don't write inputs to disk. Default: True

1.9.5 ngs_toolkit.recipes.enrichr

A helper script to run enrichment analysis using the Enrichr API on a gene set.

```
usage: python -m ngs_toolkit.recipes.enrichr [-h] [-a MAX_ATTEMPTS]
                                             [--no-overwrite]
                                             input_file output_file
```

Positional Arguments

input_file	Input file with gene names.
output_file	Output CSV file with results.

Named Arguments

-a, --max-attempts	Maximum attempts to retry the API before giving up. Default: 5
--no-overwrite	Whether results should not be overwritten if existing. Default: True

1.9.6 ngs_toolkit.recipes.generate_project

A helper script to generate synthetic data for a project in PEP format.

```
usage: python -m ngs_toolkit.recipes.generate_project [-h]
                                                      [--output-dir OUTPUT_DIR]
                                                      [--project-name PROJECT_NAME]
                                                      [--organism ORGANISM]
                                                      [--genome-assembly GENOME_
↪ASSEMBLY]
                                                      [--data-type DATA_TYPE]
                                                      [--n-factors N_FACTORS]
```

(continues on next page)

(continued from previous page)

```
                                                    [--only-metadata ONLY_METADATA]
                                                    [--sample-input-files SAMPLE_
↪INPUT_FILES]
                                                    [--debug]
```

Named Arguments

--output-dir	
--project-name	Default: "test_project"
--organism	Default: "human"
--genome-assembly	Default: "hg38"
--data-type	Default: "ATAC-seq"
--n-factors	Default: 1
--only-metadata	Default: False
--sample-input-files	Default: False
--debug	Default: False

1.9.7 ngs_toolkit.recipes.lola

A helper script to run Location Overlap Analysis (LOLA) of a single region set in various sets of region-based annotations.

```
usage: python -m ngs_toolkit.recipes.lola [-h] [--no-overwrite] [-c CPUS]
                                          bed_file universe_file output_folder
                                          genome
```

Positional Arguments

bed_file	BED file with query set regions.
universe_file	BED file with universe where the query set came from.
output_folder	Output directory for produced files.
genome	Genome assembly of the region set.

Named Arguments

--no-overwrite	Don't overwrite existing output files.
	Default: True
-c, --cpus	Number of CPUS/threads to use for analysis.

1.9.8 ngs_toolkit.recipes.merge_signal

Merge signal from various ATAC-seq or ChIP-seq samples given a set of attributes to group samples by.

It produces merged BAM and bigWig files for all signal in the samples but is also capable of producing this for nucleosomal/nucleosomal free signal based on fragment length distribution if data is paired-end sequenced. This signal may optionally be normalized for each group. It is also capable of parallelizing work in jobs.

```
usage: python -m ngs_toolkit.recipes.merge_signal [-h] [-a ATTRIBUTES] [-q]
                                                  [-j] [--cpus CPUS]
                                                  [--normalize] [--nucleosome]
                                                  [--overwrite]
                                                  [-o OUTPUT_DIR] [-d]
                                                  config_file
```

Positional Arguments

config_file YAML project configuration file.

Named Arguments

-a, --attributes Attributes to merge samples by. A comma-delimited string with no spaces. By default will use values in the project config *group_attributes*.

-q, --pass-qc Whether only samples with a 'pass_qc' value of '1' in the annotation sheet should be used.

Default: False

-j, --as-jobs Whether jobs should be created for each sample, or it should run in serial mode.

Default: False

--cpus CPUs/Threads to use per job if *–as-jobs* is on.

Default: 8

--normalize Whether tracks should be normalized to total sequenced depth.

Default: False

--nucleosome Whether to produce nucleosome/nucleosome-free signal files.

Default: False

--overwrite Whether to overwrite existing files.

Default: False

-o, --output-dir Directory for output files. Default is 'merged' under the project root directory.

Default: "merged"

-d, --dry-run Whether to do everything except running commands.

Default: False

1.9.9 ngs_toolkit.recipes.region_enrichment

A helper script to run enrichment analysis of a single region set in region-based set of annotations.

```
usage: python -m ngs_toolkit.recipes.region_enrichment [-h]
                                                       [--output-file OUTPUT_FILE]
                                                       [--overwrite]
                                                       bed_file pep
```

Positional Arguments

bed_file	BED file with regions.
pep	The analysis' PEP config file.

Named Arguments

--output-file	Output file. Default: "region_type_enrichment.csv"
--overwrite	Don't overwrite any existing directory or file. Default: False

1.9.10 ngs_toolkit.recipes.region_set_frip

Compute fraction of reads in peaks (FRiP) based on a consensus set of regions derived from several samples.

```
usage: python -m ngs_toolkit.recipes.region_set_frip [-h] [-d DATA_TYPE]
                                                     [-n NAME] [-r REGION_SET]
                                                     [-q] [-j] [-o OUTPUT_DIR]
                                                     [-s]
                                                     config_file
```

Positional Arguments

config_file	YAML project configuration file.

Named Arguments

-d, --data-type	Data types to perform analysis on. Will be done separately for each.
-n, --analysis-name	Name of analysis. Will be the prefix of output_files. By default it will be the name of the Project given in the YAML configuration.
-r, --region-set	BED file with region set derived from several samples or Oracle region set. If unset, will try to get the *sites* attribute of an existing analysis object if existing, otherwise will create a region set from the peaks of all samples.
-q, --pass-qc	Whether only samples with a 'pass_qc' value of '1' in the annotation sheet should be used. Default: False

-j, --as-jobs	Whether jobs should be created for each sample, or it should run in serial mode. Default: False
-o, --results-output	Directory for analysis output files. Default is 'results' under the project roort directory. Default: "results"
-s, --strict	Whether to throw an error in case files cannot be created or not. Default: False

1.10 API

The great flexibility of `ngs_toolkit` comes from the ability to compose workflows using the API.

It provides a rich but abstract *Analysis* object and implements various modules building on it depending on the data type.

In addition, the *general* module contains several analysis-independent methods and the *utils* module provides low-level functions of general use.

1.10.1 ngs_toolkit.analysis

class ngs_toolkit.analysis.**Analysis**(*name=None*, *from_pep=False*, *from_pickle=False*, *root_dir=None*, *data_dir='data'*, *results_dir='results'*, *prj=None*, *samples=None*, *subset_to_data_type=True*, ***kwargs*)

Generic class holding functions and data from a typical NGS analysis.

Other modules implement classes inheriting from this that in general contain data type-specific functions (e.g. *ATACSeqAnalysis* has a *get_consensus_sites()* function to generate a peak consensus map).

Objects of this type can be used to store data (e.g. dataframes), variables (e.g. paths to files or configurations) and can easily be filled with existing data using *load_data()* for cross-environment portability, or serialized (saved to a file as a pickle object) for rapid loading in the same environment. See the *to_pickle()*, *from_pickle()* and *update()* functions for this.

Parameters

- **name** (`str`, optional) – Name of the analysis.

  Defaults to "analysis".

- **from_pep** (`str`, optional) – PEP configuration file to initialize analysis from. The analysis will adopt as much attributes from the PEP as possible but keyword arguments passed at initialization will still have priority.

  Defaults to `None` (no PEP used).

- **from_pickle** (`str`, optional) – Pickle file of an existing serialized analysis object from which the analysis should be loaded.

  Defaults to `None` (will not load from pickle).

- **root_dir** (`str`, optional) – Base directory for the project.

  Defaults to current directory or to what is specified in PEP if *from_pep*.

- **data_dir** (str, optional) – Directory containing processed data (e.g. by looper) that will be input to the analysis. This is in principle not required.

  Defaults to "data".

- **results_dir** (str, optional) – Directory to contain outputs produced by the analysis.

  Defaults to "results".

- **prj** (peppy.Project, optional) – A peppy.Project object that this analysis is tied to.

  Defaults to None.

- **samples** (list, optional) – List of peppy.Sample objects that this analysis is tied to.

  Defaults to None.

- **subset_to_data_type** (bool, optional) – Whether to keep only samples that match the data type of the analysis.

  Defaults to True.

- **kwargs** (dict, optional) – Additional keyword arguments will simply be stored as object attributes.

from_pep(*pep_config*, ***kwargs*)

Create a peppy.Project from a PEP configuration file and associate is with the analysis.

Parameters **pep_config** (str) – PEP configuration file.

Variables **prj** (peppy.Project) – Project object from given PEP configuration file.

update(*pickle_file=None*)

Update all of the object"s attributes with the attributes from a serialized object (ie object stored in a file) object.

Parameters **pickle_file** (str, optional) – Pickle file to load.

Defaults to the analysis' pickle_file.

set_organism_genome()

Attempt to derive the analysis' organism and genome assembly by inspecting the same attributes of its samples.

Variables

- **organism** (str) – Organism of the analysis if all samples agree in these attributes.
- **genome** (str) – Genome assembly of the analysis if all samples agree in these attributes.

set_project_attributes(*overwrite=True*, *subset_to_data_type=True*)

Set Analysis object attributes samples, sample_attributes and group_atrributes to the values in the associated Project object if existing.

Parameters

- **overwrite** (bool, optional) – Whether to overwrite attribute values if existing.

  Defaults to True.

- **subset_to_data_type** (bool, optional) – Whether to subset samples and comparison_table to entries of same data_type as analysis.

  Defaults to True.

Variables

- **samples** (list) – List of peppy.Samples if contained in the PEP configuration.
- **sample_attributes** (list) – Sample attributes if specified in the PEP configuration.
- **group_attributes** (list) – Groups attributes if specified in the PEP configuration.
- **comparison_table** (pandas.DataFrame) – Comparison table if specified in the PEP configuration.

set_samples_input_files(*overwrite=True*)
Add input file values to sample objects dependent on data type. These are specified in the ngs_toolkit configuration file under sample_input_files:<data type>:<attribute>.

Parameters **overwrite** (bool, optional) – Whether to overwrite attribute values if existing.

Defaults to True.

to_pickle(*timestamp=False*)
Serialize object (ie save to disk) to pickle format.

Parameters **timestamp** (bool, optional) – Whether to timestamp the file.

Defaults to False.

from_pickle(*pickle_file=None*)
Load object from pickle file.

Parameters **pickle_file** (str, optional) – Pickle file to load.

Default is the object's attribute pickle_file.

Returns The analysis serialized in the pickle file.

Return type Analysis

get_sample_annotation(*attributes=None*, *samples=None*)
Get dataframe annotation of sample attributes.

Variables

- **attributes** (None, optional) – Attributes to include.

  Defaults to the union of sample_attributes and group_attributes in Analysis.
- **samples** (None, optional) – Samples to subset.

  Defaults to all samples in Analysis.

Returns Dataframe with requested attributes (columns) for each sample (rows).

Return type pandas.DataFrame

load_data(*output_map=None*, *only_these_keys=None*, *prefix='{results_dir}/{name}'*, *permissive=True*)
Load the output files of the major functions of the Analysis.

Parameters

- **output_map** (dict) – Dictionary with {attribute_name: (file_path, kwargs)} to load the files. The kwargs in the tuple will be passed to pandas.read_csv().

  Default is the required to read the keys in only_these_keys.
- **only_these_keys** (list, optional) – Iterable of analysis attributes to load up. Possible attributes:

  – "matrix_raw"

- "matrix_norm"
- "matrix_features"
- "differential_results"
- "differential_enrichment"

Default is all of the above.

- **prefix** (str, optional) – String prefix of files to load. Variables in curly braces will be formated with attributes of analysis.

  Default is "{results_dir}/{name}".

- **permissive** (bool, optional) – Whether an error should be ignored if reading a file causes IOError.

  Default is True.

Variables **<various>** (pandas.DataFrame) – Dataframes holding the respective data, available as attributes described in the only_these_keys parameter.

Raises **IOError** – If not permissive and a file is not found.

record_output_file(*file_name*, *name='analysis'*, *dump_yaml=True*, *output_yaml='{root_dir}/{name}.analysis_record.yaml'*)

Record an analysis output.

Will also write all records to a YAML file and call *Analysis.generate_report* if specified in general configuration.

Parameters

- **file_name** (str) – Filename of output to report.
- **name** (str, optional) – Name of the output to report.

  Defaults to "analysis".

- **dump_yaml** (bool, optional) – Whether to dump records to yaml file.

  Defaults to True.

- **output_yaml** (str, optional) – YAML file to dump records to. Will be formated with Analysis variables.

  Defaults to "{root_dir}/{name}.analysis_record.yaml".

Variables **output_files** (list) – Appends a tuple of (name, file_name) to output_files.

generate_report(*output_html='{root_dir}/{name}.analysis_report.html'*, *template=None*, *pip_versions=True*)

Record an analysis output.

Parameters

- **output_html** (str) – Filename of output to report.

  Defaults to "{root_dir}/{name}.analysis_report.html".

- **template** (None, optional) – Name of the output to report.

  Default is the HTML template distributed with ngs-toolkit.

- **pip_versions** (bool, optional) – Whether the versions of Python packages should be included in the report by using pip freeze.

  Default is True.

set_matrix(*matrix_name*, *csv_file*, *prefix='{results_dir}/{name}'*, ***kwargs*)
Set an existing CSV file as the value of the analysis' matrix.

Parameters

- **matrix_name** (str) – The attribute name of the matrix.

  Options are "matrix_raw" and "matrix_norm".

- **csv_file** (str) – Path to valid CSV file to be used as matrix. Assumes header and index column. Customize additional overwriding options to read CSV by passing kwargs.

- **prefix** (str, optional) – String prefix of paths to save files. Variables in curly braces will be formated with attributes of analysis.

  Defaults to "{results_dir}/{name}".

- ****kwargs** (dict) – Additional keyword arguments will be passed to pandas.read_csv.

Variables **matrix_name** (pandas.DataFrame) – An attribute named *matrix_name* holding the respecive matrix.

get_resources(*steps=['blacklist', 'tss', 'genomic_context'], organism=None, genome_assembly=None, output_dir=None, overwrite=False*)
Get genome-centric resources used by several *ngs_toolkit* analysis functions.

Parameters

- **steps** (list, optional) – What kind of annotations to get. Options are:
  - "genome": Genome sequence (2bit format)
  - "blacklist": Locations of blacklisted regions for genome
  - "tss": Locations of gene"s TSSs
  - "genomic_context": Genomic context of genome

  Defaults to ["blacklist", "tss", "genomic_context"].

- **organism** (str, optional) – Organism to get for. Currently supported are "human" and "mouse".

  Defaults to analysis' own organism.

- **genome_assembly** (str, optional) – Genome assembly to get for. Currently supported are "hg19", "hg38" and "mm10".

  Defaults to analysis' own genome assembly.

- **output_dir** (str, optional) – Directory to save results to.

  Defaults to the value of preferences:root_reference_dir in the configuration, if that is not set, to a directory called "reference" in the analysis root directory.

- **overwrite** (bool, optional) – Whether existing files should be overwritten by new ones. Otherwise they will be kept and no action is made.

  Defaults to False.

Returns Dictionary with keys same as the options as steps, containing paths to the requested files. The values of the 'genome' step are also a dictionary with keys "2bit" and "fasta" for each file type respectively.

Return type dict

normalize_rpm(*matrix='matrix_raw'*, *samples=None*, *mult_factor=1000000.0*, *log_transform=True*, *pseudocount=1*, *save=True*, *assign=True*)

Normalization of matrix of (n_features, n_samples) by total in each sample.

Parameters

- **matrix** (str, optional) – Attribute name of matrix to normalize.

  Defaults to "matrix_raw".

- **samples** (list, optional) – Iterable of peppy.Sample objects to restrict matrix to.

  Defaults to all samples in matrix.

- **mult_factor** (float, optional) – A constant to multiply values for.

  Defaults to 1e6.

- **log_transform** (bool, optional) – Whether to log transform values or not.

  Defaults to True.

- **pseudocount** ({int, float}, optional) – A constant to add to values.

  Defaults to 1.

- **save** (bool, optional) – Whether to write normalized DataFrame to disk.

  Defaults to True.

- **assign** (bool, optional) – Whether to assign the normalized DataFrame to matrix_norm.

  Defaults to True.

Variables

- **matrix_norm** (pandas.DataFrame) – If assign is True, a pandas.DataFrame normalized with respective method.
- **norm_method** (str) – If assign, it is the name of method used to normalize: "rpm".

Returns Normalized pandas DataFrame.

Return type pandas.DataFrame

normalize_quantiles(*matrix='matrix_raw'*, *samples=None*, *implementation='Python'*, *log_transform=True*, *pseudocount=1*, *save=True*, *assign=True*)

Quantile normalization of matrix of (n_features, n_samples).

Parameters

- **matrix** (str) – Attribute name of matrix to normalize.

  Defaults to "matrix_raw".

- **samples** (list, optional) – Iterable of peppy.Sample objects to restrict matrix to.

  Defaults to all in matrix.

- **implementation** (str, optional) – One of Python or R. Dictates which implementation is to be used. The R implementation comes from the *preprocessCore* package, and the Python one is from https://github.com/ShawnLYU/Quantile_Normalize. They give very similar results.

  Default is "Python".

- **log_transform** (bool, optional) – Whether to log transform values or not.

  Default is True.

- **pseudocount** (*float, optional*) – A constant to add before log transformation.

  Default is 1.

- **save** (bool, optional) – Whether to write normalized DataFrame to disk.

  Default is True.

- **assign** (bool, optional) – Whether to assign the normalized DataFrame to an attribute *matrix_norm*.

  Default is True.

Variables

- **matrix_norm** (pandas.DataFrame) – If *assign* is True, a pandas DataFrame normalized with respective method.
- **norm_method** (str) – If assign, it is the name of method used to normalize: "quantile".

Returns Normalized pandas DataFrame.

Return type pandas.DataFrame

normalize_median(*matrix='matrix_raw'*, *samples=None*, *function=<function nanmedian>*, *fillna=True*, *save=True*, *assign=True*)

Normalization of matrices of (n_features, n_samples) by subtracting the median from each sample/feature. Most appopriate for CNV data.

Parameters

- **matrix** (str, optional) – Attribute name of dictionary of matrices to normalize.

  Defaults to "matrix_raw".

- **samples** (list, optional) – Samples to restrict analysis to.

  Defaults to all samples in Analysis object.

- **function** (*function, optional*) – An alternative function to calculate across samples. Data will be subtracted by this.

  Defaults to numpy.nanmedian.

- **fillna** (bool, optional) – Whether to fill NaN with zero.

  Defaults to True.

- **save** (bool, optional) – Whether results should be saved to disc.

  Defaults to True.

- **assign** (bool, optional) – Whether to assign the normalized DataFrame to an attribute *matrix_norm*.

  Default is True.

Variables

- **matrix_norm** (pandas.DataFrame) – If *assign* is True, a pandas DataFrame normalized with respective method.
- **norm_method** (str) – If assign, it is the name of method used to normalize: "median".

Returns Normalized pandas DataFrame.

Return type pandas.DataFrame

normalize_pca(*pc*, *matrix='matrix_raw'*, *samples=None*, *save=True*, *assign=True*, ***kwargs*)

Normalization of a matrix by subtracting the contribution of Principal Component *pc* from each sample/feature.

Parameters

- **pc** (int) – Principal Component to remove. 1-based.
- **matrix** (str, optional) – Attribute name of dictionary of matrices to normalize.

  Defaults to "matrix_raw".
- **samples** (list, optional) – Samples to restrict analysis to.

  Defaults to all samples.
- **save** (bool, optional) – Whether results should be saved to disc.

  Defaults to True.
- **assign** (bool, optional) – Whether to assign the normalized DataFrame to an attribute *matrix_norm*.

  Default is True.
- ****kwargs** (dict, optional) – Additional keyword arguments will be passed to *ngs_toolkit.general.subtract_principal_component*.

Variables

- **matrix_norm** (pandas.DataFrame) – If *assign* is True, a pandas DataFrame normalized with respective method.
- **norm_method** (str) – If assign, it is the name of method used to normalize: "pca".

Returns Normalized pandas DataFrame.

Return type pandas.DataFrame

normalize_vst(*matrix='matrix_raw'*, *samples=None*, *save=True*, *assign=True*, ***kwargs*)

Normalization of a matrix using Variance Stabilization Transformation (VST) method from DESeq2.

Parameters

- **matrix** (str, optional) – Attribute name of dictionary of matrices to normalize.

  Defaults to "matrix_raw".
- **samples** (list, optional) – Samples to restrict analysis to.

  Defaults to all samples.
- **save** (bool, optional) – Whether results should be saved to disc.

  Defaults to True.

- **assign** (bool, optional) – Whether to assign the normalized DataFrame to an attribute *matrix_norm*.

  Default is True.

- ****kwargs** (dict) – Additional keywork arguments will be passed to *DESeq2::varianceStabilizingTransformation*.

Variables

- **matrix_norm** (pandas.DataFrame) – If *assign* is True, a DataFrame normalized with VST method.
- **norm_method** (str) – If assign, it is the name of method used to normalize: "vst".

Returns Normalized pandas DataFrame.

Return type pandas.DataFrame

normalize(*method='quantile'*, *matrix='matrix_raw'*, *samples=None*, *save=True*, *assign=True*, ***kwargs*)

Normalization of matrix of (n_features, n_samples).

Parameters

- **method** (str, optional) –

  Normalization method to apply. One of:

  - rpm: Reads per million normalization (RPM).
  - quantile: Quantile normalization and log2 transformation.
  - **cqn: Conditional quantile normalization (uses cqn R package).** Only available for ATAC-seq.
  - **median: Substraction of median per feature.** Only useful for CNV.
  - **pca: Subtraction of Principal Component from matrix.** Requires which PC to subtract. pc must be passed as kwarg.

  Defaults to "quantile".

- **matrix** (str, optional) – Attribute name of matrix to normalize.

  Defaults to "matrix_raw".

- **samples** (list, optional) – Iterable of peppy.Sample objects to restrict matrix to.

  Default is all samples in matrix.

- **save** (bool, optional) – Whether to write normalized DataFrame to disk.

  Defaults to True.

- **assign** (bool, optional) – Whether to assign the normalized DataFrame to an attribute *matrix_norm*.

  Default is True.

- ****kwargs** (dict) – Additional keyword arguments will be passed to the respective normalization function.

Variables

- **matrix_norm** (pandas.DataFrame) – If *assign* is True, a pandas DataFrame normalized with respective method.
- **norm_method** (str) – If assign, it is the method used to normalize.

Returns Normalized pandas DataFrame.

Return type pandas.DataFrame

remove_factor_from_matrix(*factor*, *method='combat'*, *covariates=None*, *matrix='matrix_norm'*, *samples=None*, *save=True*, *assign=True*, *make_positive=True*)

Remove an annotated factor from a matrix.

Parameters

- **factor** (str) – The name of the factor to remove from matrix.
- **method** (str) – The method to use to remove the factor.

  Default is "combat".
- **covariates** (list) – Covariates to consider when removing factor. These will be kept in the data.
- **matrix** (*{str, pandas.DataFrame}*) – The name of the attribute with the matrix or a DataFrame.

  Defaults to "matrix_norm".
- **samples** (list) – Iterable of peppy.Sample objects to restrict matrix to.

  Default (None is passed) is not to subset matrix.
- **save** (bool, optional) – Whether to write normalized DataFrame to disk.

  Defaults to True.
- **assign** (bool) – Whether to assign the result to "matrix_norm".

  Defaults to True.
- **make_positive** (bool) – Whether to make resulting matrix non-negative. Not implemented yet.

  Defaults to True.

Returns Requested matrix (dataframe).

Return type pandas.DataFrame

get_matrix(*matrix*, *samples=None*)

Get a matrix that is an attribute of self subsetted for the requested samples.

Parameters

- **matrix** (*{str, pandas.DataFrame}*) – The name of the attribute with the matrix or a DataFrame already.
- **samples** (list) – Iterable of peppy.Sample objects to restrict matrix to.

  Default (None is passed) is not to subset matrix.

Returns Requested matrix (dataframe).

Return type pandas.DataFrame

get_matrix_stats(*matrix='matrix_raw'*, *samples=None*, *save=True*, *output_prefix='stats_per_feature'*, *assign=True*)

Gets a matrix of feature-wise (ie for every gene or region) statistics such across samples such as mean, variance, deviation, dispersion and amplitude.

Parameters

- **matrix** (str) – Attribute name of matrix to normalize.

  Defaults to "matrix_raw".

- **samples** (list [peppy.Sample]) – Subset of samples to use.

  Defaults to all in analysis.

- **save** (bool, optional) – Whether to write the annotated DataFrame to disk.

  Default is True.

- **output_prefix** (str, optional) – Prefix to add to output file when save is True.

  Default is "matrix_features".

- **assign** (bool, optional) – Whether to assign the annoatated DataFrame to "matrix_features".

  Default is True.

Returns Statistics for each feature.

Return type pandas.DataFrame

Variables **stats** (pandas.DataFrame) – A DataFrame with statistics for each feature.

annotate_features(*samples=None*, *matrix='matrix_norm'*, *feature_tables=None*, *permissive=True*, *save=True*, *assign=True*, *output_prefix='matrix_features'*)

Annotates analysis features (regions/genes) by aggregating annotations per feature (genomic context, chromatin state, gene annotations and statistics) if present and relevant depending on the data type of the Analysis.

The numeric matrix to be used is specified in *matrix*. If any two two annotation dataframes have equally named columns (e.g. chrom, start, end), the value of the first is kept.

Parameters

- **samples** (list) – Iterable of peppy.Sample objects to restrict matrix to. Calculated metrics will be restricted to these samples.

  Defaults to all in analysis (the matrix will not be subsetted).

- **matrix** (str) – Attribute name of matrix to annotate.

  Defaults to "matrix_norm".

- **feature_tables** (list) – Attribute names of dataframes used to annotate the numeric dataframe.

  Default is ["gene_annotation", "region_annotation", "chrom_state_annotation", "support", "stats"] for ATAC-seq and ChIP-seq and ["stats"] for all others.

- **permissive** (bool) – Whether DataFrames that do not exist should be simply skipped or an error will be thrown.

  Defaults to True.

- **save** (bool, optional) – Whether to write the annotated DataFrame to disk.

  Default is True.

- **assign** (bool, optional) – Whether to assign the annoatated DataFrame to "matrix_features".

  Default is True.

- **output_prefix** (str, optional) – Prefix to add to output file when save is True.

  Default is "matrix_features".

Raises **AttributeError** – If not *permissive* a required DataFrame does not exist as an object attribute.

Variables **matrix_features** (pandas.DataFrame) – A pandas DataFrame containing annotations of the region features.

annotate_samples(*matrix='matrix_norm'*, *attributes=None*, *numerical_attributes=None*, *save=False*, *assign=False*)

Annotate matrix (n_features, n_samples) with sample metadata (creates MultiIndex on columns). Numerical attributes can be pass as a iterable to numerical_attributes to be converted.

Parameters

- **matrix** (str, optional) – Attribute name of matrix to annotate.

  Defaults to "matrix_norm".

- **attributes** (list, optional) – Desired attributes to be annotated.

  Defaults to all attributes in the original sample annotation sheet of the analysis' Project.

- **numerical_attributes** (list, optional) – Attributes which are numeric even though they might be so in the samples" attributes. Will attempt to convert values to numeric.

- **save** (bool, optional) – Whether to write normalized DataFrame to disk.

  Default is True.

- **assign** (bool, optional) – Whether to assign the normalized DataFrame to "matrix_norm".

  Default is True.

Returns Annotated dataframe with requested sample attributes.

Return type pandas.DataFrame

Variables **matrix_norm** (pandas.DataFrame) – A pandas DataFrame with MultiIndex column index containing the sample's attributes specified.

annotate_matrix(***kwargs*)

Convinience function to create dataframes annotated with feature and samples attributes.

Simply calls Analysis.annotate_features and analysis.annotate_samples.

Parameters **kwargs** (dict) – Additional keyword arguments are passed to the above mentioned functions.

get_level_colors(*index=None*, *matrix='matrix_norm'*, *levels=None*, *pallete='tab20'*, *uniform_cmap='plasma'*, *diverging_cmap='RdYlBu_r'*, *nan_color=(0.662745, 0.662745, 0.662745, 1.0)*, *as_dataframe=False*)

Get tuples of floats representing a colour for a sample in a given variable in a dataframe"s index (particularly useful with MultiIndex dataframes).

If given, will use the provieded index argument, otherwise, the columns and its levels of an attribute of self named matrix. levels can be passed to subset the levels of the index.

Will try to guess if each variable is categorical or numerical and return either colours from a colour pallete or a cmap, respectively with null values set to nan_color (a 4-value tuple of floats).

Parameters

- **index** (*pandas.Index, optional*) – Pandas Index to use.

  Default is to use the column Index of the provided `matrix`.

- **matrix** (`str`, optional) – Name of analysis attribute containing a dataframe with pandas.MultiIndex columns to use.

  Default is to use the provided `index`.

- **levels** (`list`, optional) – Levels of multiindex to restrict to.

  Defaults to all in index under use.

- **pallete** (`str`, optional) – Name of matplotlib color palete to use with categorical levels. See matplotlib.org/examples/color/colormaps_reference.html.

  Defaults to "tab20".

- **{uniform_cmap, diverging_cmap}** (`str`, optional) – Name of matplotlib color paletes to use with numerical levels. Uniform will be used if values in level are non-negative, while diverging if including negative. See matplotlib.org/examples/color/colormaps_reference.html.

  Defaults to "plasma" and "RdYlBu_r", respectively.

- **nan_color** (*tuple, optional*) – Color for missing (i.e. NA) values.

  Defaults to `(0.662745, 0.662745, 0.662745, 0.5)` == `grey`.

- **as_dataframe** (`bool`, optional) – Whether a dataframe should be returned.

  Defaults to `False`.

Returns Matrix of shape (level, sample) with rgb values of each of the variable. If as_dataframe, this will be a pandas.DataFrame otherwise, list of lists.

Return type {list, pandas.DataFrame}

unsupervised_analysis(*steps=['correlation', 'manifold', 'pca', 'pca_association'], matrix='matrix_norm', samples=None, attributes_to_plot=None, output_dir='{results_dir}/unsupervised_analysis_{data_type}', output_prefix='all_{var_unit_name}s', standardize_matrix=True, manifold_algorithms=['MDS', 'Isomap', 'LocallyLinearEmbedding', 'SpectralEmbedding', 'TSNE'], maniford_kwargs={}, display_corr_values=False, plot_max_pcs=4, save_additional=False, prettier_sample_names=True, rasterized=False, dpi=300, **kwargs*)

General unsupervised analysis of a matrix.

Apply unsupervised clustering, manifold learning and dimensionality reduction methods on numeric matrix. Colours and labels samples by their attributes as given in *attributes_to_plot*.

This analysis has 4 possible steps:

- **"correlation":** Pairwise sample correlation with 2 distance metrics plotted as heatmap.
- **"manifold":** Manifold learning of latent spaces for projection of samples. See here available algorithms: http://scikit-learn.org/stable/modules/classes.html#module-sklearn.manifold
- **"pca":** For PCA analysis, if *test_pc_association* is *True*, will compute association of PCs with sample attributes given in *attributes_to_plot*. For numeric attributes, the Pearson correlation will be computed and for categoriacal, a pairwise Kruskal-Wallis H-test (ANOVA).

- **"pca_association":** For PCA analysis, if *test_pc_association* is *True*, will compute association of PCs with sample attributes given in *attributes_to_plot*. For numeric attributes, the Pearson correlation will be computed and for categoriacal, a pairwise Kruskal-Wallis H-test (ANOVA).

Parameters

- **steps** (`list`, optional) – List of step keywords to be performed as described above.

  Defaults to all available.

- **matrix** (`str`, optional) – Name of analysis attribute contatining the numeric dataframe to perform analysis on. Must have a pandas.MultiIndex as column index.

  Defaults to "matrix_norm".

- **samples** (`list`, optional) – List of sample objects to restrict analysis to.

  Defaults to all in analysis.

- **attributes_to_plot** (`list`, optional) – List of attributes shared between sample groups should be plotted.

  Defaults to attributes in analysis.group_attributes.

- **output_dir** (`str`, optional) – Directory for generated files and plots.

  Defaults to "{results_dir}/unsupervised_analysis_{data_type}".

- **output_prefix** (`str`, optional) – Prefix for output files.

  Defaults to "all_regions" if data_type is ATAC-seq and "all_genes" if data_type is RNA-seq.

- **standardize_matrix** (`bool`, optional) – Whether to standardize variables in *matrix* by removing the mean and scaling to unit variance. It is not applied to the "correlation" step.

  Default is `True`.

- **manifold_algorithms** (`list`, optional) – List of manifold algorithms to use. See available algorithms here: http://scikit-learn.org/stable/modules/classes.html#module-sklearn.manifold

  Defaults to ['MDS', 'Isomap', 'LocallyLinearEmbedding', 'SpectralEmbedding', 'TSNE'],

- **maniford_kwargs** (`dict`, optional) – Dictionary of keyword arguments to pass to the algorithms in `manifold_algorithms`. Should be of the form {"algorithm_name": {"key": value}}

- **display_corr_values** (`bool`, optional) – Whether values in heatmap of sample correlations should be displayed overlaid on top of colours.

  Defaults to `False`.

- **save_additional** (`bool`, optional) – Whether additional results such as PCA projection, loadings should be saved.

  Defaults to `False`.

- **prettier_sample_names** (`bool`, optional) – Whether it should attempt to prettify sample names by removing the data type from plots.

  Defaults to `True`.

- **rasterized** (bool, optional) – Whether elements with many objects should be rasterized.

  Defaults to False.

- **dpi** (int, optional) – Definition of rasterized image in dots per inch (dpi).

  Defaults to 300.

- ****kwargs** (*optional*) – kwargs are passed to get_level_colors() and *plot_projection()*.

differential_analysis(*comparison_table=None*, *samples=None*, *covariates=None*, *filter_support=False*, *output_dir='{results_dir}/differential_analysis_{data_type}'*, *output_prefix='differential_analysis'*, *overwrite=True*, *distributed=False*, *deseq_kwargs=None*, ***kwargs*)

Perform differential regions/genes across samples that are associated with a certain trait. Currently the only implementation is with DESeq2. This implies the rpy2 library and the respective R library are installed.

Requires the R package "DESeq2" to be installed:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("DESeq2")
```

For other implementations of differential analysis see *ngs_toolkit.general.least_squares_fit* and *ngs_toolkit.general.differential_from_bivariate_fit*.

Parameters

- **comparison_table** (pandas.DataFrame) – A dataframe with "comparison_name", "comparison_side" and "sample_name", "sample_group" columns.

  Defaults to the analysis' own "comparison_table" attribute.

- **samples** (list, optional) – Samples to limit analysis to.

  Defaults to all samples in analysis object.

- **covariates** (list, optional) – Additional variables to take into account in the model fitting.

  Defaults to None.

- **filter_support** (bool, optional) – Whether features not supported in a given comparison should be removed (i.e. regions with no peaks in any sample in a comparison are not tested). Applies only to ATAC-/ChIP-seq data.

  Default is True.

- **output_dir** (str, optional) – Output directory for analysis. Variables in curly braces will be formated with attributes from analysis.

  Defaults to "{results_dir}/differential_analysis_{data_type}".

- **output_prefix** (str, optional) – Prefix for output files.

  Defaults to "differential_analysis".

- **overwrite** (bool, optional) – Whether results should be overwritten in case they already exist.

  Defaults to True.

- **distributed** (bool, optional) – Whether analysis should be distributed in a computing cluster for each comparison. Additional configuration can be passed in `kwargs`.

  Defaults to `False`.

- **deseq_kwargs** (dict, optional) – Additional keyword arguments to be passed to the *DESeq* function of DESeq2.

- **kwargs** (dict, optional) – Additional keyword arguments are passed to *submit_job()* and then to the chosen *divvy* submission template according to *computing_configuration*. Pass for example *cores=4, mem=8000, partition="longq", time="08:00:00"*.

Returns Results for all comparisons. Will be `None` if *distributed* is *True*.

Return type pandas.DataFrame

Variables **differential_results** (pandas.DataFrame) – Pandas dataframe with results.

collect_differential_analysis(*comparison_table=None*, *input_dir='{results_dir}/differential_analysis_{data_type}'*, *input_prefix='differential_analysis'*, *output_dir='{results_dir}/differential_analysis_{data_type}'*, *output_prefix='differential_analysis'*, *permissive=True*, *save=True*, *assign=True*, *overwrite=False*)

Collect results from DESeq2 differential analysis. Particularly useful when running `differential_analysis` in distributed mode.

Parameters

- **comparison_table** (pandas.DataFrame) – A dataframe with "comparison_name", "comparison_side" and "sample_name", "sample_group" columns.

  Defaults to the analysis's own "comparison_table" attribute.

- **input_dir, output_dir** (str, optional) – In-/Output directory of files. Values within curly brackets "{data_type}", will be formated with attributes from analysis.

  Defaults to "{results_dir}/differential_analysis_{data_type}".

- **input_prefix, output_prefix** (str, optional) – Prefix of the in-/output files.

  Defaults for both is "differential_analysis".

- **permissive** (bool, optional) – Whether non-existing files should be skipped or an error be thrown.

  Defaults to `True`.

- **save** (bool, optional) – Whether to save results to disk.

  Defaults to `True`.

- **assign** (bool, optional) – Whether to add results to a *differential_results* attribute.

  Defaults to `True`.

- **overwrite** (bool, optional) – Whether results should be overwritten in case they already exist.

  Defaults to `False`.

Returns Results for all comparisons. Will be `None` if `overwrite` is `False` and a results file already exists.

Return type `pandas.DataFrame`

Variables **`differential_results`** (`pandas.DataFrame`) – Pandas dataframe with results.

`plot_differential`(*steps=['distributions', 'counts', 'scatter', 'volcano', 'ma', 'stats_heatmap', 'correlation', 'heatmap'], results=None, comparison_table=None, samples=None, matrix='matrix_norm', only_comparison_samples=False, alpha=0.05, corrected_p_value=True, fold_change=None, diff_based_on_rank=False, max_rank=1000, ranking_variable='pvalue', respect_stat_thresholds=True, output_dir='{results_dir}/differential_analysis_{data_type}', output_prefix='differential_analysis', plot_each_comparison=True, mean_column='baseMean', log_fold_change_column='log2FoldChange', p_value_column='pvalue', adjusted_p_value_column='padj', comparison_column='comparison_name', rasterized=True, robust=False, feature_labels=False, group_colours=True, group_attributes=None, **kwargs*)

Plot differential features (eg chromatin region, genes) discovered with supervised group comparisons by `ngs_toolkit.general.differential_analysis`. This will plot number and direction of discovered features, scatter, MA and volcano plots for each comparison and joint heatmaps of log fold changes, normalized values or Z-scores of individual samples or groups in the differential features.

Parameters

- **steps** (`list`, optional) –

  Types of plots to make:

  - "distributions": Distribution of p-values and fold-changes
  - "counts" - Count of differential features per comparison given certain thresholds.
  - "scatter" - Scatter plots (group 1 vs group 2).
  - "volcano" - Volcano plots (log fold change vs -log p-value)
  - "ma" - MA plots (log mean vs log fold change)
  - "stats_heatmap" - Heatmap of p-values and fold-changes for comparisons.
  - "correlation" - Correlation of samples or sample groups in differential features.
  - "heatmap" - Heatmaps of samples or sample groups in differential features.

  Defaults to all of the above.

- **results** (`pandas.DataFrame`, optional) – Data frame with differential analysis results. See `ngs_toolkit.general.differential_analysis` for more information.

- **comparison_table** (`pandas.DataFrame`, optional) – Comparison table. If provided, group-wise plots will be produced.

  Defaults to the analysis' "comparison_table" attribute.

- **samples** (`list`, optional) – List of sample objects to restrict analysis to.

  Defaults to all samples in analysis.

- **matrix** (`str`, optional) – Matrix of quantification to use for plotting feature values across samples/groups.

  Defaults to "matrix_norm".

- **only_comparison_samples** (`bool`, optional) – Whether to use only samples present in the *comparison_table* and *results* table.

  Defaults to `False`.

- **alpha** (*float, optional*) – Significance level to consider a feature differential.

  Defaults to 0.05.

- **corrected_p_value** (`bool`, optional) – Whether to use a corrected p-valueto consider a feature differential.

  Defaults to `True`.

- **fold_change** (*float, optional*) – Effect size (log2 fold change) to consider a feature differential. Considers absolute values.

  Default is no log2 fold change threshold.

- **diff_based_on_rank** (`bool`, optional) – Whether a feature should be considered differential based on its rank. Use in combination with *max_rank*, *ranking_variable* and *respect_stat_thresholds*.

  Defaults to `False`.

- **max_rank** (`int`, optional) – Rank to use when using *diff_based_on_rank*.

  Defaults to 1000.

- **ranking_variable** (`str`, optional) – Which variable to use for ranking when using *diff_based_on_rank*.

  Defaults to "pvalue".

- **respect_stat_thresholds** (`bool`, optional) – Whether the statistical thresholds from *alpha* and *fold_change* should still be respected when using *diff_based_on_rank*.

  Defaults to `True`.

- **output_dir** (`str`, optional) – Directory to create output files.

  Defaults to "{results_dir}/differential_analysis_{data_type}"

- **output_prefix** (`str`, optional) – Prefix to use when creating output files.

  Defaults to "differential_analysis".

- **plot_each_comparison** (`bool`, optional) – Whether each comparison should be plotted in scatter, MA and volcano plots. Useful to turn off with many comparisons.

  Defaults to `True`.

- **mean_column** (`str`, optional) – Column in *results* data frame containing values for mean values across samples.

  Defaults to "baseMean".

- **log_fold_change_column** (`str`, optional) – Column in *results* data frame containing values for log2FoldChange values across samples.

  Defaults to "log2FoldChange".

- **p_value_column** (str, optional) – Column in *results* data frame containing values for p-values across samples.

  Defaults to "pvalue".

- **adjusted_p_value_column** (str, optional) – Column in *results* data frame containing values for adjusted p-values across samples.

  Defaults to "padj".

- **comparison_column** (str, optional) – Column in *results* data frame containing the name of the comparison.

  Defaults to "comparison_name".

- **rasterized** (bool, optional) – Whether plots with many objects should be rasterized.

  Defaults to True.

- **robust** (bool, optional) – Whether heatmap color scale ranges should be robust (using quantiles) rather than extreme values. Useful for noisy/extreme data.

  Defaults to False.

- **feature_labels** (bool, optional) – Whether features (regions/genes) should be labeled in heatmaps.

  Defaults to False.

- **group_colours** (bool, optional) – Whether groups of samples should be coloured in heatmaps.

  Defaults to True.

- **group_attributes** (list, optional) – Which variables to colour if *group_colours* if True.

  Defaults to all of analysis.group_attributes.

- ****kwargs** (dict, optional) – Additional keyword arguments will be passed to *Analysis.get_level_colors*.

differential_overlap(*differential=None*, *output_dir='{results_dir}/differential_analysis_{data_type}'*, *output_prefix='differential_analysis'*)

Visualize intersection of sets of differential regions/genes.

Parameters

- **differential** (pandas.DataFrame, optional) – DataFrame containing result of comparisons filtered for features considered as differential.

  Defaults to the differential_results attribute, subset by the object's thresholds.

- **output_dir** (str, optional) – Directory to create output files.

  Defaults to "{results_dir}/differential_analysis_{data_type}".

- **output_prefix** (str, optional) – Prefix to use when creating output files.

  Defaults to "differential_analysis".

differential_enrichment(*differential=None, output_dir='{results_dir}/differential_analysis_{data_type}/enrichments', output_prefix='differential_analysis', genome=None, steps=['region', 'lola', 'meme', 'homer', 'enrichr'], directional=True, max_diff=1000, sort_var='pvalue', distributed=False, overwrite=False*)

Perform various types of enrichment analysis given a dataframe of the results from differential analysis. Performs enrichment of gene sets (RNA-seq and ATAC-seq), genomic regions, chromatin states Location Overlap Analysis (LOLA) and TF motif enrichment (over-representation and de-novo search) (ATAC-seq only).

Parameters

- **differential** (`pandas.DataFrame`) – Data frame with differential results as produced by `differential_analysis`, but filtered by some threshold for the relevant (significant regions). Must contain a "comparison_name" column.

  Defaults to `analysis.differential_results`.

- **output_dir** (`str`, optional) – Directory to create output files.

  Defaults to "{results_dir}/differential_analysis_{data_type}".

- **output_prefix** (`str`, optional) – Prefix to use when creating output files.

  Defaults to "differential_analysis".

- **genome** (`str`, optional) – Genome assembly of the analysis.

  Defaults to Analysis's `genome` attribute.

- **steps** (`list`, optional) – Steps of the analysis to perform.

  Defaults to all possible: ["region", lola", "meme", "homer", "enrichr"].

- **directional** (`bool`, optional) – Whether enrichments should be performed in a direction-dependent way (up-regulated and down-regulated features separately). This requires a column named "log2FoldChange" to exist.

  Defaults to `True`.

- **max_diff** (`int`, optional) – Number of maximum features to perform enrichment for ranked by variable in *max_diff*.

  Defaults to 1000.

- **sort_var** (`str`, optional) – Variable to sort for when setting *max_diff*.

  Defaults to "pvalue".

- **distributed** (`bool`, optional) – Whether work should be submitted as jobs in a computing cluster.

  Defaults to `False`.

- **overwrite** (`bool`, optional) – Whether output files should be overwritten when *distributed* is `True`.

  Defaults to `False`.

Variables **enrichment_results** (`dict`) – Dictionary with keys as in *steps* and values with pandas.DataFrame of enrichment results.

collect_differential_enrichment(*steps=['region', 'lola', 'motif', 'homer', 'homer_consensus', 'enrichr'], directional=True, permissive=True, output_dir='{results_dir}/differential_analysis_{data_type}/enrichments', input_prefix='differential_analysis', output_prefix='differential_analysis', differential=None*)

Collect the results of enrichment analysis ran after a differential analysis.

Parameters

- **steps** (`list`, optional) – Steps of the enrichment analysis to collect results for.

  Defaults to ["region", "lola", "meme", "homer", "enrichr"].

- **directional** (`bool`, optional) – Whether enrichments were made in a direction-dependent way (up-regulated and down-regulated features separately). This implies a column named "direction" exists".

  Defaults to `True`.

- **differential** (`pandas.DataFrame`, optional) – Data frame with differential results to select which comparisons to collect enrichments for. Usually produced by `ngs_toolkit.general.differential_analysis`.

  Defaults to analysis's `differential_results` attributes.

- **output_dir** (`str`, optional) – Directory to create output files.

  Defaults to "{results_dir}/differential_analysis_{data_type}".

- **input_prefix, output_prefix** (`str`, optional) – File prefix of input/output files.

  Defaults to "differential_analysis".

- **permissive** (`bool`, optional) – Whether to skip non-existing files, giving a warning.

  Defaults to `True`.

Variables **enrichment_results** (`dict`) – Dictionary with keys as in `steps` and values with pandas.DataFrame of enrichment results.

plot_differential_enrichment(*steps=['region', 'lola', 'motif', 'great', 'enrichr'], plot_types=['barplots', 'scatter', 'correlation', 'heatmap'], enrichment_type=None, enrichment_table=None, direction_dependent=True, output_dir='{results_dir}/differential_analysis_{data_type}/enrichments', comp_variable='comparison_name', output_prefix='differential_analysis', rasterized=True, clustermap_metric='correlation', top_n=5, z_score=0, cmap=None*)

Make plots illustrating enrichment of features for various comparisons.

Input can be the dictionary under *analysis.enrichment_results* or a single dataframe of enrichment terms across several comparisons for a given type of enrichment. In the later case both *enrichment_table* and *enrichment_type* must be given.

Parameters

- **steps** (`list`, optional) – Types of the enrichment analysis to plot. Options are ["region", "lola", "motif", "great", "enrichr"].

  Defaults to all keys present in analysis.enrichment_results.

- **plot_types** (`list`, optional) – Types of plots to do for each enrichment type. One of ["barplot", "scatter", "correlation", "heatmap"].

  Defaults to all of the above.

- **enrichment_type** (`str`, optional) – Type of enrichment if run for a single type of enrichment. In this case *enrichment_table* must be given. One of {"region", "lola", "motif", "great", "enrichr"}.

  Default (`None`) is to run all keys present in analysis.enrichment_results.

- **enrichment_table** (`pandas.DataFrame`, optional) – Data frame with enrichment results as produced by `differential_enrichment` or `collect_differential_enrichment`. If given, *enrichment_type* must be given too.

  Default (`None`) is the dataframes in all values present in analysis.enrichment_results.

- **direction_dependent** (`bool`, optional) – Whether enrichments were made in a direction-dependent way (up-regulated and down-regulated features separately). This implies a column named "direction" exists".

  Defaults to `True`.

- **output_dir** (`str`, optional) – Directory to create output files.

  Defaults to "{results_dir}/differential_analysis_{data_type}/enrichments".

- **comp_variable** (`str`, optional) – Column defining which comparison enrichment terms belong to.

  Defaults to "comparison_name".

- **output_prefix** (`str`, optional) – Prefix to use when creating output files.

  Defaults to "differential_analysis".

- **rasterized** (`bool`, optional) – Whether or not to rasterize heatmaps for efficient plotting.

  Defaults to `True`.

- **clustermap_metric** (`str`, optional) – Distance metric to use for clustermap clustering, See https://docs.scipy.org/doc/scipy/reference/spatial.distance.html for valid values.

  Default to "correlation" (Pearson's).

- **top_n** (`int`, optional) – Top terms to use to display in plots.

  Defaults to 5.

- **z_score** (*{bool, int}, optional*) – Which dimention/axis to perform Z-score transformation for. Pass `False` to skip plotting Z-score heatmaps. Numpy/Pandas conventions are used: *0* is row-wise (in this case across comparisons) and *1* is column-wise (across terms).

  Defaults to 0.

- **cmap** (`str`, optional) – Colormap to use in heatmaps.

  Defaults to `None`.

run_full_analysis_recipe(***kwargs*)

Run the `ngs_toolkit.recipes.ngs_analysis` recipe on the current Analysis object.

Parameters **kwargs (dict) – Additional keyword arguments are passed to ngs_toolkit.recipes.ngs_analysis.main_analysis_pipeline().

1.10.2 ngs_toolkit.atacseq

class ngs_toolkit.atacseq.**ATACSeqAnalysis**(*name=None*, *from_pep=False*, *from_pickle=False*, *root_dir=None*, *data_dir='data'*, *results_dir='results'*, *prj=None*, *samples=None*, ***kwargs*)

Class to model analysis of ATAC-seq data. Inherits from the *Analysis* class.

Parameters

- **name** (str, optional) – Name of the analysis.

  Defaults to "analysis".

- **from_pep** (str, optional) – PEP configuration file to initialize analysis from. The analysis will adopt as much attributes from the PEP as possible but keyword arguments passed at initialization will still have priority.

  Defaults to None (no PEP used).

- **from_pickle** (str, optional) – Pickle file of an existing serialized analysis object from which the analysis should be loaded.

  Defaults to None (will not load from pickle).

- **root_dir** (str, optional) – Base directory for the project.

  Defaults to current directory or to what is specified in PEP if *from_pep*.

- **data_dir** (str, optional) – Directory containing processed data (e.g. by looper) that will be input to the analysis. This is in principle not required.

  Defaults to "data".

- **results_dir** (str, optional) – Directory to contain outputs produced by the analysis.

  Defaults to "results".

- **prj** (peppy.Project, optional) – A peppy.Project object that this analysis is tied to.

  Defaults to None.

- **samples** (list, optional) – List of peppy.Sample objects that this analysis is tied to.

  Defaults to None.

- **kwargs** (dict, optional) – Additional keyword arguments will be passed to parent class *Analysis*.

Examples

```
>>> from ngs_toolkit.atacseq import ATACSeqAnalysis
```

This is an example of the beginning of an ATAC-seq analysis:

```
>>> pep = "metadata/project_config.yaml"
>>> a = ATACSeqAnalysis(from_pep=pep)
>>> # Get consensus peak set from all samples
>>> a.get_consensus_sites(a.samples)
>>> # Annotate regions
>>> a.get_peak_gene_annotation()
>>> a.get_peak_genomic_location()
>>> # Get coverage values for each peak in each sample of ATAC-seq
>>> a.measure_coverage()
>>> # Normalize jointly (quantile normalization + GC correction)
>>> a.normalize(method="gc_content")
>>> # Annotate quantified peaks with previously calculated metrics and features
>>> a.annotate_features()
>>> # Annotate with sample metadata
>>> a.annotate_samples()
>>> # Save object
>>> a.to_pickle()
```

load_data(*output_map=None*, *only_these_keys=None*, *prefix='{results_dir}/{name}'*, *permissive=True*)

Load the output files of the major functions of the Analysis.

Parameters

- **output_map** (dict) – Dictionary with "attribute name": "path prefix" to load the files.
- **only_these_keys** (list, optional) – Iterable of analysis attributes to load up. Possible attributes:
  - "matrix_raw"
  - "matrix_norm"
  - "matrix_features"
  - "sites"
  - "support"
  - "nuc"
  - "coverage_gc_corrected"
  - "gene_annotation"
  - "region_annotation"
  - "region_annotation_b"
  - "chrom_state_annotation"
  - "chrom_state_annotation_b"
  - "stats"
  - "differential_results"

  Default is all of the above.
- **prefix** (str, optional) – String prefix of files to load. Variables in curly braces will be formated with attributes of analysis.

  Defaults to "{results_dir}/{name}".

- **bool** (*permissive, optional*) – Whether an error should be ignored if reading a file causes IOError.

  Default is `True`.

Variables

- **sites** (`pybedtools.bedtool.BedTool`) – Sets a *sites* variable.
- **pandas.DataFrame** – Dataframes holding the respective data, available as attributes described in the *only_these_keys* parameter.

Raises **IOError** – If not permissive and a file is not found

get_consensus_sites(*samples=None*, *region_type='summits'*, *extension=250*, *blacklist_bed=None*, *filter_chroms=None*, *permissive=False*, *save=True*, *assign=True*, ***kwargs*)

Get consensus (union) of enriched sites (peaks) across samples. There are two modes possible, defined by the value of `region_type`:

- peaks: simple union of all sites;
- summits: peak summits are extended by `extension` and a union is made.

Parameters

- **samples** (`list`) – Iterable of `peppy.Sample` objects to restrict to. Must have a `peaks` attribute set.

  Defaults to all samples in the analysis (`samples` attribute).
- **region_type** (`str`) – The type of region to use to create the consensus region set - one of "summits" or "peaks". If "summits", peak summits will be extended by `extension` before union. If "peaks", sample peaks will be used with no modification prior to union.

  Default is "summits".
- **extension** (`int`) – Amount to extend peaks summits by in both directions.

  Default is 250.
- **blacklist_bed** ({`False`, `str`}) – Either `False` or a path to a BED file with genomic positions to exclude from consensus peak set.

  Default is to use a blacklist file for the analysis `genome`.
- **filter_chroms** ({`list`, `str`}) – A list of chromosomes to filter out or a string with a pattern to match to exclude chromosomes. Uses Pandas string methods `pandas.Series.str.match`. Pass for example *".*_.*|chrM"* to filter out chromosomes with a "_" character and a "chrM" chromosome.

  Default is not to filter anything.
- **permissive** (`bool`) – Whether Samples that which `region_type` attribute file does not exist should be simply skipped or an error thrown.

  Default is `True`.
- ****kwargs** – Not used. Provided for compatibility with `ngs_toolkit.ChIPSeqAnalysis` class.

Raises

- **ValueError** – If not `permissive` and either the `peaks` or `summits` file of a sample is not readable, or if `permissive` but none of the samples has an existing file.
- **AttributeError** – If analysis does not have `organism` and `genome` attributes.

Variables **sites** (`pybedtools.bedtool.BedTool`) – Sets a `sites` variable with the consensus peak set.

Returns **sites** – The consensus peak set.

Return type `pybedtools.bedtool.BedTool`

set_consensus_sites(*bed_file*, *overwrite=True*)

Set consensus (union) sites across samples given a BED file.

Parameters

- **bed_file** (`str`) – BED file to use as consensus sites.
- **overwrite** (`bool`) – Whether a possibly existing file with a consensus peak set for this analysis should be overwritten in disk.

Variables **sites** (`BedTool`) – Sets a *sites* variable with consensus peak set.

calculate_peak_support(*samples=None*, *region_type='summits'*, *permissive=False*, *comparison_table=None*, *peak_dir=None*)

Count number of called peaks per sample in the consensus region set. In addition calculate a measure of peak support (or ubiquitouness) by observing the ratio of samples containing a peak overlapping each region.

Parameters

- **samples** (`list`) – Iterable of `peppy.Sample` objects to restrict to. Must have a `peaks` attribute set.

  Defaults to all samples in the analysis (`samples` attribute).
- **region_type** (`str`) – The type of region to use to create the consensus region set. One of "summits" or "peaks". If *summits*, peak summits will be extended by `extension` before union. Otherwise sample peaks will be used with no modification.

  Default is "summits".
- **permissive** (`bool`) – Whether Samples that which *region_type* attribute file does not exist should be simply skipped or an error thrown.
- **comparison_table** (`pandas.DataFrame`) – Not used. Provided for compatibility with ChIPSeqAnalysis class.
- **peak_dir** (`str`) – Not used. Provided for compatibility with ChIPSeqAnalysis class.

Raises **IOError** – If not `permissive` and either the `peaks` or `summits` file of a sample is not readable. Or if `permissive` but none of the samples has an existing file.

Variables **support** (`pandas.DataFrame`) – A dataframe with counts of peaks overlapping each feature of consensus set.

get_supported_peaks(*samples=None*, ***kwargs*)

Get mask of sites with 0 support in the given samples. Requires support matrix produced by *ngs_toolkit.atacseq.ATACSeqAnalysis.calculate_peak_support*.

Parameters

- **samples** (list) – Iterable of peppy.Sample objects to restrict to.
- ****kwargs** – Not used. Provided for compatibility with ChIPSeqAnalysis class.

Returns Boolean Pandas Series with sites with at least one of the given samples having a peak called.

Return type pd.Series

measure_coverage(*samples=None*, *sites=None*, *save=True*, *assign=True*, *peak_set_name='peak_set'*, *output_file='{results_dir}/{name}.matrix_raw.csv'*, *permissive=False*, *distributed=False*, *overwrite=True*, ***kwargs*)

Measure read coverage (counts) of each sample in each region in consensus sites. Uses parallel computing using the parmap library. However, for many samples (hundreds), parallelization in a computing cluster is possible with the *distributed* option.

Parameters

- **samples** (list) – Iterable of peppy.Sample objects to restrict to. Must have a aligned_filtered_bam attribute set.

  Defaults to all samples in the analysis (samples attribute).

- **sites** ({pybedtools.bedtool.BedTool, pandas.DataFrame, str}) – Sites in the genome to quantify, usually a pybedtools.bedtool.BedTool from *ngs_toolkit.atacseq.ATACSeqAnalysis.get_consensus_sites*. If a DataFrame, will try to convert to BED format assuming first three columns are chr,start,end. If a string assumes a path to a BED file.

  Defaults to sites attribute of analysis object.

- **save** (bool) – Whether to save to disk the coverage matrix with filename output_file.

  Default is True.

- **assign** (bool) – Whether to assign the matrix to an attribute named coverage.

  Default is True.

- **peak_set_name** (bool) – Suffix to files containing coverage of distributed is True.

  Defaults to "peak_set".

- **output_file** (str) – A path to a CSV file with coverage output.

  Default is "{results_dir}/{name}.raw_coverage.csv".

- **permissive** (bool) – Whether Samples for which region_type attribute file does not exist should be simply skipped or an error thrown.

  Default is False.

- **distributed** (bool) – Whether it should be run as jobs for each sample separately in parallel. Currently only implemented for a SLURM cluster.

  Default is False.

- **overwrite** (bool) – Whether to overwrite existing files if distributed is True.

  Default is True.

- ****kwargs** (dict) – Additional keyword arguments will be passed to *ngs_toolkit.utils.submit_job* if *distributed* is True, and on to a divvy submission template. Pass for example: computing_configuration="slurm", jobname="job", cores=2, mem=8000, partition="longq".

Raises **IOError** – If not `permissive` and the 'aligned_filtered_bam' file attribute of a sample is not readable. Or if `permissive` but none of the samples has an existing file.

Variables **matrix_raw** (pandas.DataFrame) – The dataframe of raw coverage values (counts) of shape (n_features, m_samples).

Returns Pandas DataFrame with read counts of shape (n_sites, m_samples).

Return type pandas.DataFrame

collect_coverage(*samples=None*, *save=True*, *assign=True*, *output_file=None*, *permissive=False*, *peak_set_name='peak_set'*, *fast_and_unsafe=False*)

Collect read coverage (counts) of each sample in each region in consensus sites from existing files. Useful after runnning analysis.measure_coverage() in distributed mode.

Parameters

- **samples** (list) – Iterable of `peppy.Sample` objects to restrict to. If not provided (*None* is passed) if will default to all samples in the analysis (`samples` attribute).
- **save** (bool) – Whether to save to disk the coverage matrix with filename `output_file`.
- **assign** (bool) – Whether to assign the matrix to an attribute of self named `coverage`.
- **output_file** (str) – A path to a CSV file with coverage output.

  Default is "{results_dir}/{name}.raw_coverage.csv".
- **permissive** (bool) – Whether Samples without an existing coverage file does not exist should be simply skipped or an error thrown.
- **peak_set_name** (bool) – Suffix to files containing coverage. Defaults to "peak_set".
- **fast_and_unsafe** (bool) – Whether to use a faster but unsafer method to concatenate the data. If the order of all rows in all samples is the same then the result should be the same. The default, slower method assures that all rows are matched and is therefore slower.

  Defaults to `False`.

Raises **IOError** – If not `permissive` and the coverage file of a sample is not readable or is empty. Or if `permissive` but none of the samples has an existing file or are empty.

Variables **matrix_raw** (pandas.DataFrame) – The dataframe of raw coverage values (counts) of shape (n_features, m_samples).

Returns Pandas DataFrame with read counts of shape (n_sites, m_samples).

Return type pandas.DataFrame

get_peak_gccontent_length(*bed_file=None*, *fasta_file=None*)

Get length and GC content of features in region set.

bed_file [str] A BED file with regions to calculate GC content on. Must be a 3-column BED! If not provided the calculation will be for the analysis *sites* attribute.

genome [str] Genome assembly.

fasta_file [str] Fasta file of *genome*. Preferably indexed. If not given, will try to download.

Variables

- **nuc** – DataFrame with nucleotide content and length of each region.
- **nuc** (pandas.DataFrame) – Dataframe with length and GC-content of each feature.

Returns DataFrame with nucleotide content and length of each region.

Return type pandas.DataFrame

normalize_cqn(*matrix='matrix_raw'*, *samples=None*, *save=True*, *assign=True*)

Conditional quantile normalization (CQN) of a matrix. It uses GC content and length of regulatory elements as covariates.

Requires the R package "cqn" to be installed:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("cqn")
```

Parameters

- **matrix** (str) – Attribute name of matrix to normalize.

  Defaults to "matrix_raw".
- **samples** (list) – Iterable of peppy.Sample objects to restrict matrix to.

  Defaults to all samples in analysis.
- **save** (bool) – Whether to write normalized DataFrame to disk.

  Default is True.
- **assign** (bool) – Whether to assign the normalized DataFrame to an attribute *matrix_norm*.

  Default is True.

Variables

- **matrix_norm** (pandas.DataFrame) – If *assign*, the dataframe with normalized values.
- **norm_method** (str) – If assign, it is the name of method used to normalize: "cqn".

get_peak_gene_annotation(*tss_file=None*, *max_dist=100000*, *save=True*, *output_prefix=''*, *assign=True*)

Annotates peaks with closest gene. The annotation reference can either be given in the *tss_file* parameter but if ommited, it will be fetched if analysis has *genome* and *organism* attributes. A dataframe with each feature's distance to the nearest gene is also saved.

Parameters

- **tss_file** (str, optional) – A valid BED file where the name field (4th column) identifies the gene and the strand column (6th column). Other fields will not be used.

Default is to get gene position annotations.

- **max_dist** (int, optional) – Maximum absolute distance allowed to perform associations. Regions with no genes within the range will have NaN values.

  Default is 100000.

- **save** (bool, optional) – Whether to write the annotated DataFrame to disk.

  Default is True.

- **output_prefix** (str, optional) – Prefix to add to output file when save is True.

  Default is "" (empty string).

- **assign** (bool, optional) – Whether to assign the DataFrames to *Attributes*.

  Default is True.

Variables

- **gene_annotation** (pandas.DataFrame) – A pandas DataFrame containing the genome annotations of the region features. If a feature overlaps more than one gene, the two gene values will be concatenated with a comma.
- **closest_tss_distances** (pandas.DataFrame) – A pandas DataFrame containing unique region->gene associations. In contrast to gene_annotation dataframe, this contains one row per region->gene assignment.

Returns A dataframe with genes annotated for the peak set.

Return type pandas.DataFrame

get_peak_genomic_location(*genomic_context_file=None*, *save=True*, *output_prefix=''*, *assign=True*)

Annotates a consensus peak set (sites attribute of analysis) with their genomic context. The genomic context is mostly gene-centric, which includes overlap with gene promoters, UTRs, exons, introns and remaining intergenic space.

If no reference genomic annotation file is given (genomic_context_file kwarg), it will use the ngs_toolkit.general.get_genomic_context function to get such data. For more customization of the annotations, use that function directly and pass the output file to this function.

Parameters

- **genomic_context_file** (str) – A 4 column BED file (chrom, start, end, feature), where feature is a string with the type of region. If not provided will be get with the get_genomic_context function.
- **save** (bool, optional) – Whether to write the annotated DataFrame to disk.

  Default is True.

- **output_prefix** (str, optional) – Prefix to add to output file when save is True.

  Default is "" (empty string).

- **assign** (bool, optional) – Whether to assign the DataFrames to *Attributes*.

  Default is True.

Variables

- **region_annotation_b** (*region_annotation,*) – A DataFrame with the genome annotations of the region features or genome background.

- **region_annotation_b_mapping** (*region_annotation_mapping,*) – A DataFrame with one row for each chromatin state-region mapping or genome background.

Returns The genomic context annotation for the peak set.

Return type pandas.DataFrame

get_peak_chromatin_state(*chrom_state_file*, *frac=0.2*, *save=True*, *output_prefix=''*, *assign=True*)

Annotates a consensus peak set (`sites` attribute of analysis) with their chromatin state context. This would be given, for example by a chromatin state segmentation file from projects such as Roadmap Epigenomics.

See examples of such files for various cell types/assemblies in this website (the "dense.bed.gz" files are optimal).

Parameters

- **chrom_state_file** (`str`) – A 4 column BED file (chrom, start, end, feature), where feature is a string with the type of region. Additional columns are ignored.
- **frac** (*float*) – Minimal fraction of region to overlap with a feature.

  Defaults to 0.2.
- **save** (`bool`, optional) – Whether to write the annotated DataFrame to disk.

  Default is `True`.
- **output_prefix** (`str`, optional) – Prefix to add to output file when save is True.

  Default is "" (empty string).
- **assign** (`bool`, optional) – Whether to assign the DataFrames to *Attributes*.

  Default is `True`.

Returns The chromatin state annotation for the peak set.

Return type pandas.DataFrame

Variables

- **chrom_state_annotation_b** (*chrom_state_annotation,*) – A DataFrame with the chromatin state annotations of the region features or of the genome background.
- **chrom_state_annotation_b_mapping**(*chrom_state_annotation_mapping,*) – A DataFrame with one row for each chromatin state-region mapping or for the genome background.

get_sex_chrom_ratio(*matrix='matrix_norm'*, *sex_chroms=['chrX', 'chrY']*, *output_dir='{results_dir}'*, *output_prefix='sex_chrom_ratio'*, *plot=True*)

Get ratio of signal between sex chromosomes. Useful to quickly assign sex to samples.

Parameters

- **matrix** (`pandas.DataFrame`, optional) – Matrix to use. Defaults to *matrix_norm*.
- **sex_chroms** (`list`, optional) – Names of the two sex chromosomes to use.
- **output_dir** (`str`, optional) – Directory to write output to.
- **output_prefix** (`str`, optional) – String to prefix output with.

- **plot** (bool, optional) – Whether to produce illustrative plots.

Returns Ratio of sex chromosomes defined as *sex_chroms[1] - sex_chroms[0]*.

Return type pd.Series

get_gene_level_matrix(*matrix='matrix_norm'*, *reduce_func=<function mean>*, *assign=True*, *save=True*, *output_file='{results_dir}/{name}.gene_coverage.csv'*)

Get gene-level measurements of coverage.

Requires a 'gene_annotation' or 'closest_tss_distances' attribute to be set containing a mapping between the index of *matrix* and genes (produced from *get_peak_gene_annotation*).

Parameters

- **matrix** (str, optional) – Quantification matrix to use (e.g. 'matrix_raw' or 'matrix_norm')

  Default is "matrix_norm".

- **reduce_func** (*func*) – Function to apply to reduce values.

  Default is mean.

- **assign** (bool) – Whether to assign the matrix to an attribute of self named *matrix_gene*.

  Default is True.

- **save** (bool) – Whether to save to disk the coverage matrix with filename *output_file*.

  Default is True.

- **output_file** (str) – Path to save a CSV file with coverage output if *save* is *True*.

  Default is *self.results_dir/self.name + ".raw_coverage.csv"*.

Returns Coverage values reduced per gene.

Return type pandas.DataFrame

Variables **matrix_gene** (pandas.DataFrame) – Coverage values reduced per gene.

get_gene_level_changes(*differential_results=None*, *reduce_func=<function mean>*)

Redcuce changes in regulatory elements to gene-level by aggregating across regulatory elements. Requires a 'gene_annotation' attribute to be set containing a mapping between the index of *matrix* and genes (produced from *get_peak_gene_annotation*).

Parameters

- **differential_results** (pandas.DataFrame) – Matrix with differential results to use. Default is a 'differential_results' attribute of self.
- **reduce_func** (*func*) – Function to apply to reduce values. Default is mean

Returns Changes in chromatin accessibility (log2FoldChanges) reduced per gene.

Return type pandas.DataFrame

plot_peak_characteristics(*samples=None*, *by_attribute=None*, *genome_space=3000000000.0*, *output_dir='{results_dir}/peak_characteristics'*, *output_prefix='{name}'*)

Several diagnostic plots on the analysis' consensus peak set and the sample's signal on them.

Provides plots with samples grouped *by_attribute* if given (a string or a list of strings).

Parameters

- **samples** (list, optional) – List of samples to restrict analysis to.
- **by_attribute** (*{str, list}, optional*) – Attribute or list of sample attributes to groupby samples by when plotting. This is done in addition to the plots with individual values per sample.
- **genome_space** (int) – Length of genome.

  Defaults to 3e9 basepairs (human genome).
- **output_dir** (str) – Directory to output files. Will be formated with variables from Analysis.

  Defaults to "peak_characteristics" under the Analysis "results_dir".
- **output_prefix** (str) – Prefix to add to output files.

  Defaults to the Analysis' name.

plot_raw_coverage(*samples=None*, *by_attribute=None*)

Diagnostic plots on the Sample's signal. Provides plots with Samples grouped *by_attribute* if given (a string or a list of strings).

Parameters

- **samples** (list) – List of peppy.Samples objects to use for plotting.
- **by_attribute** (str, optional) – Attribute of samples to group by. Values will be aggregated across samples by that attribute.

region_context_enrichment(*regions*, *steps=['genomic_region', 'chromatin_state']*, *background='region_set'*, *prefix='region_type_enrichment'*, *output_dir='{results_dir}'*)

Characterize a subset of the regions (e.g. differential regions) in terms of their genomic context.

Parameters

- **regions** ({list, pandas.DataFrame, pandas.Index}) – Subset of regions of interest to analysis. Must be a subset of the universe (i.e. sites attribute).
- **steps** (list, optional) – Steps of enrichment to perform. Defaults to all available: 'genomic_region' and 'chromatin_state'.
- **background** (str, optional) – Which set to consider as backgroud. Options are:
  - "region_set": the consensus region_set of the analysis
  - "genome": a randomized set of size as region_set across the genome
- **prefix** (str, optional) – Prefix for saved files.

  Default is "region_type_enrichment".
- **output_dir** (str, optional) – Directory to write results to. Can be formatted with Analysis attributes.

  Default is "{results_dir}".

Returns Enrichment results

Return type pandas.DataFrame

characterize_regions_function(*differential, output_dir, prefix, universe_file=None, run=True, genome=None, steps=['region', 'lola', 'meme', 'homer', 'enrichr']*)

Performs a range of functional enrichments of a set of regions given in `differential` (a dataframe which is typically a subset of an annotated coverage dataframe). Will extract regions, their underlying sequence, associated genes, perform enrichment of genomic regions, chromatin states against a background, motif enrichment, location overlap analysis (LOLA), and gene set enrichment (using the Enrichr API).

This requires several programs and R libraries:

- MEME suite (AME)
- HOMER suite (findMotifsGenome.pl)
- LOLA (R library)

Additionally, some genome-specific databases are needed to run these programs.

Parameters

- **differential** (`pandas.DataFrame`) – Results of differential analysis for a given comparison of interest.
- **output_dir** (`str`) – Directory to output results to.
- **prefix** (`str`) – Prefix to use for output files.
- **universe_file** (`str`, optional) – Path to BED file with set of universe regions where differential were selected from.

  Default is `sites` attribute of Analysis.
- **run** (`bool`, optional) – Whether to run enrichment commands now or to simply prepare the input files for it. Default is `True`.
- **genome** (`str`, optional) – Genome assembly of analysis. Default is genome genome assembly of analysis (`genome` attribute).
- **steps** (`list`, optional) – Which steps of the analysis to perform. Default is all: [‘region’, ‘lola’, ‘meme’, ‘homer’, ‘enrichr’].

1.10.3 ngs_toolkit.chipseq

class ngs_toolkit.chipseq.**ChIPSeqAnalysis**(*name=None, from_pep=False, from_pickle=False, root_dir=None, data_dir='data', results_dir='results', prj=None, samples=None, **kwargs*)

Class to model analysis of ChIP-seq data. Inherits from the *ATACSeqAnalysis* class.

Parameters

- **name** (`str`, optional) – Name of the analysis.

  Defaults to “analysis”.
- **from_pep** (`str`, optional) – PEP configuration file to initialize analysis from. The analysis will adopt as much attributes from the PEP as possible but keyword arguments passed at initialization will still have priority.

  Defaults to `None` (no PEP used).
- **from_pickle** (`str`, optional) – Pickle file of an existing serialized analysis object from which the analysis should be loaded.

Defaults to None (will not load from pickle).

- **root_dir** (str, optional) – Base directory for the project.

  Defaults to current directory or to what is specified in PEP if *from_pep*.

- **data_dir** (str, optional) – Directory containing processed data (e.g. by looper) that will be input to the analysis. This is in principle not required.

  Defaults to "data".

- **results_dir** (str, optional) – Directory to contain outputs produced by the analysis.

  Defaults to "results".

- **prj** (peppy.Project, optional) – A peppy.Project object that this analysis is tied to.

  Defaults to None.

- **samples** (list, optional) – List of peppy.Sample objects that this analysis is tied to.

  Defaults to None.

- **kwargs** (dict, optional) – Additional keyword arguments will be passed to parent class *ATACSeqAnalysis*.

call_peaks_from_comparisons(*comparison_table=None*, *output_dir='{results_dir}/chipseq_peaks'*, *permissive=True*, *overwrite=True*, *distributed=True*)

Call peaks for ChIP-seq samples using an annotation of which samples belong in each comparison and which samples represent signal or background.

Parameters

- **comparison_table** (pandas.DataFrame) – Comparison table with the following required columns: "comparison_name", "sample_name", "comparison_side", "sample_group".

  Defaults to analysis' own *comparison_table*.

- **output_dir** (str) – Parent directory where peaks will be created.

  Will be created if does not exist.

- **permissive** (bool) – If incomplete/incoherent comparisons should be skipped or an error should be thrown.

  Default is True.

- **overwrite** (bool) – If incomplete/incoherent comparisons should be skipped or an error should be thrown.

  Default is True.

- **distributed** (bool) – Whether peak calling should be run in serial or in distributed mode as jobs.

  Default is True.

Raises **ValueError** – If not *permissive* and incomplete/incoherent comparisons are detected.

filter_peaks(*comparison_table=None*, *filter_bed=None*, *peaks_dir='{results_dir}/chipseq_peaks'*)

Filter peak calls for various comparisons for entries that do not overlap another BED file.

Parameters

- **comparison_table** (pandas.DataFrame, optional) – Comparison table with the following required columns: "comparison_name", "sample_name", "comparison_side", "sample_group".

  Defaults to analysis' own *comparison_table*.

- **filter_bed** (str) – BED file with entries to filter out from the BED files of each comparison.

  Defaults to the set of Blacklisted regions from the analysis' genome. In that case it will be fetched if not present.

- **peaks_dir** (str) – Parent directory where peak calls for each comparison exist. Will be created if does not exist.

  Defaults to "{results_dir}/chipseq_peaks".

Raises **AttributeError** – If *filter_bed* is not given and failes to be retrieved.

summarize_peaks_from_comparisons(*comparison_table=None*, *output_dir='{results_dir}/chipseq_peaks'*, *filtered=True*, *permissive=True*)

Call peaks for ChIP-seq samples using an annotation of which samples belong in each comparison and which samples represent signal or background.

Parameters

- **comparison_table** (pandas.DataFrame, optional) – Comparison table with the following required columns: "comparison_name", "sample_name", "comparison_side", "sample_group".

  Defaults to analysis' own *comparison_table*.

- **output_dir** (str) – Parent directory where peaks will be created. Will be created if does not exist.

- **permissive** (bool) – If incomplete/incoherent comparisons should be skipped or an error should be thrown.

Raises **ValueError** – Will be raised if not *permissive* and incomplete/incoherent comparisons are detected.

get_consensus_sites(*samples=None*, *region_type='summits'*, *extension=250*, *blacklist_bed=None*, *filter_chroms=True*, *permissive=False*, *save=True*, *assign=True*, ***kwargs*)

Get consensus (union) of enriched sites (peaks) across all comparisons. There are two modes possible, defined by the value of region_type:

- peaks: simple union of all sites;
- summits: peak summits are extended by extension and a union is made.

For ChIP-seq, the comparison_table keyword argument or a comparison_table attribute set is required. Peaks/summits will be aggregated for the peaks called in each sample comparison.

Parameters

- **samples** (list) – Iterable of peppy.Sample objects to restrict to. Must have a peaks attribute set.

  Defaults to all samples in the analysis (samples attribute).

- **region_type** (str) – The type of region to use to create the consensus region set - one of "summits" or "peaks". If "summits", peak summits will be extended

by `extension` before union. If "peaks", sample peaks will be used with no modification prior to union.

Default is "summits".

- **extension** (`int`) – Amount to extend peaks summits by in both directions.

  Default is 250.

- **blacklist_bed** ({`False`, `str`}) – Either `False` or a path to a BED file with genomic positions to exclude from consensus peak set.

  Default is to use a blacklist file for the analysis `genome`.

- **filter_chroms** ({`list`, `str`}) – A list of chromosomes to filter out or a string with a pattern to match to exclude chromosomes. Uses Pandas string methods `pandas.Series.str.match`. Pass for example *'.*_.*|chrM'* to filter out chromosomes with a "_" character and a "chrM" chromosome.

  Default is not to filter anything.

- **permissive** (`bool`) – Whether Samples that which `region_type` attribute file does not exist should be simply skipped or an error thrown.

- **comparison_table** (`pandas.DataFrame`, optional) – DataFrame with signal/background combinations used to call peaks. Part of kwargs.

  Defaults to analysis own `comparison_table`.

- **peak_dir** (`str`, optional) – Path to peaks output directory. Part of kwargs.

  Defaults to "{analysis.results_dir}/chipseq_peaks".

Variables **sites** (`pybedtools.BedTool`) – Bedtool with consensus sites.

calculate_peak_support (*samples=None*, *region_type='summits'*, *permissive=False*, *comparison_table=None*, *peak_dir='{results_dir}/chipseq_peaks'*)

Calculate a measure of support for each region in peak set (i.e. ratio of samples containing a peak overlapping region in union set of peaks).

Parameters

- **comparison_table** (`pandas.DataFrame`, optional) – DataFrame with signal/background combinations used to call peaks

  Defaults to analysis' own *comparison_table*.

- **peak_dir** (`str`, optional) – Path to peaks output directory. Defaults to {analysis.results_dir}/chipseq_peaks

- **samples** (`list`) – Not used. Provided for compatibility with ATACSeqAnalysis class.

- **region_type** (`str`) – Not used. Provided for compatibility with ATACSeqAnalysis class.

- **permissive** (`bool`) – Not used. Provided for compatibility with ATACSeqAnalysis class.

Variables **support** (`pandas.DataFrame`) – DataFrame with signal/background combinations used to call peaks

get_supported_peaks (*samples=None*, ***kwargs*)

Get mask of sites with 0 support in the given samples. Requires support matrix produced by *ngs_toolkit.atacseq.ATACSeqAnalysis.calculate_peak_support*.

Parameters

- **samples** (list) – Not used. Provided for compatibility with ATACSeqAnalysis class.
- **comparisons** (list) – Iterable of comparison names to restrict to. Must match name of comparisons used in comparison_table.

Returns Boolean Pandas Series with sites with at least one of the given samples having a peak called.

Return type pd.Series

normalize_by_background(*comparison_table=None*, *reduction_func=<function mean>*, *comparison_func=<ufunc 'subtract'>*, *by_group=False*, *matrix='matrix_norm'*, *samples=None*)

Normalize values in matrix by background samples in a comparison-specific way as specified in *comparison_table*.

The background samples will be pooled by the *reduction_func* and their values wil be removed from the signal samples using the *comparison_func*.

Parameters

- **comparison_table** (pandas.DataFrame) – Table with comparisons from which peaks were called.

  Defaults to analysis' *comparison_table*.
- **reduction_func** (*func*) – Function to reduce the region to gene values to.

  Defaults to numpy.mean.
- **comparison_func** (*func*) – Function to use for normalization of signal samples against background samples. You can also try for example numpy.divide.

  Defaults to numpy.subtract.
- **by_group** (bool) – Whether output should be by group (True) or for each sample (False).

  Default is False.
- **matrix** ({pandas.DataFrame, str, optional}) – Name of attribute or pandas DataFrame to use.

  Defaults to "matrix_norm".
- **samples** (list, optional) – Subset of samples to consider.

  Defaults to all samples in analysis.

Returns Dataframe with values normalized by background samples.

Return type pandas.DataFrame

1.10.4 ngs_toolkit.cnv

class ngs_toolkit.cnv.**CNVAnalysis**(*name=None*, *from_pep=False*, *from_pickle=False*, *root_dir=None*, *data_dir='data'*, *results_dir='results'*, *prj=None*, *samples=None*, ***kwargs*)

Class to model analysis of CNV data. Inherits from the *Analysis* class.

Parameters

- **name** (`str`, optional) – Name of the analysis.

  Defaults to "analysis".

- **from_pep** (`str`, optional) – PEP configuration file to initialize analysis from. The analysis will adopt as much attributes from the PEP as possible but keyword arguments passed at initialization will still have priority.

  Defaults to `None` (no PEP used).

- **from_pickle** (`str`, optional) – Pickle file of an existing serialized analysis object from which the analysis should be loaded.

  Defaults to `None` (will not load from pickle).

- **root_dir** (`str`, optional) – Base directory for the project.

  Defaults to current directory or to what is specified in PEP if *from_pep*.

- **data_dir** (`str`, optional) – Directory containing processed data (e.g. by looper) that will be input to the analysis. This is in principle not required.

  Defaults to "data".

- **results_dir** (`str`, optional) – Directory to contain outputs produced by the analysis.

  Defaults to "results".

- **prj** (`peppy.Project`, optional) – A `peppy.Project` object that this analysis is tied to.

  Defaults to `None`.

- **samples** (`list`, optional) – List of `peppy.Sample` objects that this analysis is tied to.

  Defaults to `None`.

- **kwargs** (`dict`, optional) – Additional keyword arguments will be passed to parent class *Analysis*.

Examples

```
>>> from ngs_toolkit.cnv import CNVAnalysis
```

This is an example of a CNV analysis:

```
>>> pep = "metadata/project_config.yaml"
>>> a = CNVAnalysis(from_pep=pep)
>>> # Get consensus peak set from all samples
>>> a.get_cnv_data()
>>> # Normalize
>>> a.normalize(method="median")
>>> # Segmentation
>>> a.segment_genome()
>>> # General plots
>>> a.plot_all_data()
>>> a.plot_segmentation_stats()
>>> # Unsupervised analysis
>>> a.unsupervised_analysis()
>>> # Save object
>>> a.to_pickle()
```

load_data(*output_map=None*, *only_these_keys=None*, *resolutions=None*, *prefix='{results_dir}/{name}'*, *permissive=True*)
Load the output files of the major functions of the Analysis.

Parameters

- **output_map** (dict) – Dictionary with {attribute_name: (file_path, kwargs)} to load the files. The kwargs in the tuple will be passed to `pandas.read_csv()`.

  Defaults to the required to read the keys in `only_these_keys`.
- **only_these_keys** (list, optional) – Iterable of analysis attributes to load up. Possible attributes:
  - "matrix_raw"
  - "matrix_norm"
  - "matrix_features"
  - "differential_results"

  Defaults to all of the above.
- **resolutions** (list) – List of resolution strings to get data for.

  Defaults to value of `resolutions` attribute of Analysis.
- **prefix** (str, optional) – String prefix of files to load. Variables in curly braces will be formated with attributes of analysis.

  Defaults to "{results_dir}/{name}".
- **permissive** (bool, optional) – Whether an error should be ignored if reading a file causes IOError.

  Default is `True`.

Variables **pandas.DataFrame** – Dataframes holding the respective data, available as attributes described in the *only_these_keys* parameter.

Raises **IOError** – If not permissive and a file is not found

get_cnv_data(*resolutions=None*, *samples=None*, *save=True*, *assign=True*, *permissive=False*)
Load CNV data from ATAC-seq CNV pipeline and create CNV matrix at various resolutions.

Parameters

- **resolutions** (list, optional) – Resolutions of analysis. Defaults to resolutions in Analysis object.
- **samples** (list, optional) – Samples to restrict analysis to. Defaults to samples in Analysis object.
- **save** (bool, optional) – Whether results should be saved to disc. Defaults to True
- **assign** (bool, optional) – Whether results should be assigned to an attribute in the Analsyis object. Defaults to True
- **permissive** (bool, optional) – Whether missing files should be allowed. Defaults to False

Returns Dictionary with CNV matrices for each resolution.

Return type dict

Raises **IOError** – If not permissive and input files can't be read.

Variables **matrix** (dict) – Sets a *matrix* dictionary with CNV matrices for each resolution.

normalize(*method='median'*, *matrix='matrix_raw'*, *samples=None*, *save=True*, *assign=True*, ***kwargs*)

Normalization of dictionary of matrices with (n_features, n_samples).

Parameters

- **resolutions** (list, optional) – Resolutions of analysis. Defaults to resolutions in Analysis object.
- **method** (str) – Normalization method to apply.

  Defaults to "median".
- **matrix** (str, optional) – Attribute name of dictionary of matrices to normalize. Defaults to *matrix_raw*.
- **samples** (list) – Iterable of peppy.Sample objects to restrict matrix to. Default is all in analysis.
- **save** (bool, optional) – Whether results should be saved to disc. Defaults to True
- **assign** (bool, optional) – Whether results should be assigned to an attribute in the Analsyis object. Defaults to True
- **kwargs** (dict, optional) – Additional kwargs are passed to the respective normalization method.

Returns Dictionary with normalized CNV matrices for each resolution.

Return type dict

Variables **matrix_norm** (dict) – Sets a *matrix_norm* dictionary with CNV matrices for each resolution.

plot_all_data(*matrix='matrix_norm'*, *resolutions=None*, *samples=None*, *output_dir=None*, *output_prefix='{analysis_name}.all_data'*, *robust=True*, *vmin=None*, *vmax=None*, *rasterized=True*, *dpi=300*, *sample_labels=True*)

Visualize CNV data genome-wide using heatmaps. Will be done independently for each specified resolution.

Parameters

- **matrix** (str, optional) – Attribute name of dictionary of matrices to normalize. Defaults to *matrix_norm*.
- **resolutions** (list, optional) – Resolutions of analysis. Defaults to resolutions in Analysis object.
- **samples** (list, optional) – Samples to restrict analysis to. Defaults to samples in Analysis object.
- **output_dir** (str, optional) – Output directory. Defaults to Analysis results directory.
- **output_prefix** (str, optional) – Prefix to add to plots. Defaults to "{analysis_name}.all_data"
- **robust** (bool, optional) – Whether to scale the color scale robustly (to quantiles rather than extremes). Defaults to True
- **vmin** (*float, optional*) – Minimum value of color scale.
- **vmax** (*float, optional*) – Maximum value of color scale. Defaults to None

- **rasterized** (bool, optional) – Whether to rasterize main heatmap. Defaults to True
- **dpi** (int, optional) – DPI resolution of rasterized image. Defaults to 300
- **sample_labels** (bool, optional) – Whether to label samples with their name. Defaults to True

plot_stats_per_chromosome(*matrix='matrix_norm'*, *resolutions=None*, *samples=None*, *output_dir='{results_dir}'*, *output_prefix='{analysis_name}.all_data'*, *robust=True*, *rasterized=True*, *dpi=300*, *sample_labels=True*)

Visualize mean and variation of CNV data for each chromosome using heatmaps. Will be done independently for each specified resolution. Will also be done once for all chromosomes and another time without sex chromosomes.

Parameters

- **matrix** (str, optional) – Attribute name of dictionary of matrices to normalize. Defaults to *matrix_norm*.
- **resolutions** (list, optional) – Resolutions of analysis. Defaults to resolutions in Analysis object.
- **samples** (list, optional) – Samples to restrict analysis to. Defaults to samples in Analysis object.
- **output_dir** (str, optional) – Output directory. Defaults to Analysis results directory.
- **output_prefix** (str, optional) – Prefix to add to plots. Defaults to "{analysis_name}.all_data"
- **robust** (bool, optional) – Whether to scale the color scale robustly (to quantiles rather than extremes). Defaults to True
- **rasterized** (bool, optional) – Whether to rasterize main heatmap. Defaults to True
- **dpi** (int, optional) – DPI resolution of rasterized image. Defaults to 300
- **sample_labels** (bool, optional) – Whether to label samples with their name. Defaults to True

segment_genome(*matrix='matrix_norm'*, *resolutions=None*, *samples=None*, *save=True*, *assign=True*)

Segment CNV data to create calls of significant deviations. Will be done independently for each specified resolution.

Requires the R package "DNAcopy" to be installed:

```
>>> source('http://bioconductor.org/biocLite.R')
>>> biocLite('DNAcopy')
```

Parameters

- **matrix** (str, optional) – Attribute name of dictionary of matrices to segment. Defaults to *matrix_norm*.
- **resolutions** (list, optional) – Resolutions of analysis. Defaults to resolutions in Analysis object.

- **samples** (list, optional) – Samples to restrict analysis to. Defaults to samples in Analysis object.
- **save** (bool, optional) – Whether results should be saved to disc. Defaults to True
- **assign** (bool, optional) – Whether results should be assigned to an attribute in the Analsyis object. Defaults to True

Returns Dictionary with segmentation for each resolution.

Return type dict

Variables **segmentation** (dict) – Dictionary with CNV matrices for each resolution.

annotate_with_chrom_bands(*segmentation=None*, *resolutions=None*, *save=True*, *assign=True*)

Annotate segmentation with chromosome bands and overlaping genes. Will be done independently for each specified resolution.

Parameters

- **segmentation** (str, optional) – Attribute name of dictionary of segmentation results. Defaults to *segmentation*.
- **resolutions** (list, optional) – Resolutions of analysis. Defaults to resolutions in Analysis object.
- **samples** (list, optional) – Samples to restrict analysis to. Defaults to samples in Analysis object.
- **save** (bool, optional) – Whether results should be saved to disc. Defaults to True
- **assign** (bool, optional) – Whether results should be assigned to an attribute in the Analsyis object. Defaults to True

Returns Dictionary with annotated segmentation for each resolution.

Return type dict

Variables **segmentation_annot** (dict) – Dictionary with CNV matrices for each resolution.

plot_segmentation_stats(*segmentation=None*, *resolutions=None*, *per_sample=False*, *output_dir='{results_dir}/segmentation'*, *output_prefix='{resolution}.segmentation_metrics'*)

Visualize distribution of statistics of CNV data segmentation. Will be done independently for each specified resolution.

Parameters

- **segmentation** (str, optional) – Dictionary of segmentation results. Defaults to *segmentation*.
- **resolutions** (list, optional) – Resolutions of analysis. Defaults to resolutions in Analysis object.
- **per_sample** (bool, optional) – Whether plots should be made for each sample too. Defaults to False
- **output_dir** (str, optional) – Output directory.
- **output_prefix** (str, optional) – Prefix to add to plots. Defaults to "{resolution}.segmentation_metrics"

ngs_toolkit.cnv.**all_to_igv**(*matrix*, *output_prefix*, ***kwargs*)

Convert dictionary of DataFrame with CNV data in several resolutions to IGV format.

Parameters

- **matrix** (pandas.DataFrame) – DataFrame with CNV data to convert.
- **output_prefix** (*str*) – Prefix to add to plots.
- ****kwargs** (dict, optional) – Additional parameters will be passed to ngs_toolkit.cnv.to_igv

Returns Dictionary of CNV data in IGV format for each resolution.

Return type dict

ngs_toolkit.cnv.**to_igv**(*matrix*, *output_file=None*, *save=True*, *view_limits=(-2, 2)*)

Convert DataFrame with CNV data to IGV format.

Parameters

- **matrix** (pandas.DataFrame) – DataFrame with CNV data to convert.
- **output_file** (*str, optional*) – Output file.

  Required if *save* is True.
- **save** (bool, optional) – Whether results should be saved to disc.

  Defaults to True.
- **view_limits** (*tuple, optional*) – Extreme values (min, max) of color scale used to visualize in IGV.

  Defaults to (-2, 2).

Returns CNV data in IGV format.

Return type pandas.DataFrame

Raises **ValueError:** – If *save* is True but *output_file* is None.

1.10.5 ngs_toolkit.rnaseq

class ngs_toolkit.rnaseq.**RNASeqAnalysis**(*name=None*, *from_pep=False*, *from_pickle=False*, *root_dir=None*, *data_dir='data'*, *results_dir='results'*, *prj=None*, *samples=None*, ***kwargs*)

Class to model analysis of RNA-seq data. Inherits from the *Analysis* class.

Parameters

- **name** (str, optional) – Name of the analysis.

  Defaults to "analysis".
- **from_pep** (str, optional) – PEP configuration file to initialize analysis from. The analysis will adopt as much attributes from the PEP as possible but keyword arguments passed at initialization will still have priority.

  Defaults to None (no PEP used).
- **from_pickle** (str, optional) – Pickle file of an existing serialized analysis object from which the analysis should be loaded.

  Defaults to None (will not load from pickle).

- **root_dir** (str, optional) – Base directory for the project.

  Defaults to current directory or to what is specified in PEP if *from_pep*.

- **data_dir** (str, optional) – Directory containing processed data (e.g. by looper) that will be input to the analysis. This is in principle not required.

  Defaults to "data".

- **results_dir** (str, optional) – Directory to contain outputs produced by the analysis.

  Defaults to "results".

- **prj** (peppy.Project, optional) – A peppy.Project object that this analysis is tied to.

  Defaults to None.

- **samples** (list, optional) – List of peppy.Sample objects that this analysis is tied to.

  Defaults to None.

- **kwargs** (dict, optional) – Additional keyword arguments will be passed to parent class *Analysis*.

collect_bitseq_output(*samples=None*, *permissive=True*, *expression_type='counts'*)

Collect gene expression (read counts, transcript-level) output from Bitseq into expression matrix for *samples*.

collect_esat_output(*samples=None*, *permissive=True*)

Collect gene expression (read counts, gene-level) output from ESAT into expression matrix for *samples*.

get_gene_expression(*expression_type='counts'*, *expression_level='gene'*, *reduction_func=<built-in function max>*, *quantification_prog='bitseq'*, *samples=None*, *save=True*, *assign=True*, *output_file=None*, *permissive=False*, *species=None*, *ensembl_version=None*)

Collect gene expression (read counts per transcript or gene) for all samples.

If *expression_level* is "gene", then, transcripts will be reduced per gene ID using *reduction_func* (defaults to *max*) and features will be named with gene symbols.

Parameters

- **expression_type** (str, optional) – Type of expression quantification to get. One of "counts" or "rpkm".

  Defaults to "counts".

- **expression_level** (str, optional) – Type of expression quantification to get. One of "transcript" or "gene".

  Defaults to "gene".

- **reduction_func** (*func, optional*) – Function to reduce gene expression between transcript and gene if *expression_level* is "gene".

  Defaults to *max*.

- **quantification_prog** (str, optional) – Name of program used to produce the quantification of gene expression. One of "bitseq", "htseq" or "esat".

  Defaults to "bitseq".

- **samples** (*list[peppy.Sample], optional*) – Subset of samples to get expression for.

  Defaults to all in analysis.

- **save** (bool, optional) – Whether to save output as CSV.

  Default is None.

- **assign** (bool, optional) – Whether to assign output to *matrix_raw*.

  Default is None.

- **output_file** (str, optional) – Path of resulting file if *save* is *True*.

  Defaults to "{results_dir}/{name}.matrix_raw.csv".

- **permissive** (bool, optional) – Whether to skip samples with non-existing gene expression quantification.

  Default is *False*.

- **species** (str, optional) – Ensembl species name (e.g. "hsapiens", "mmusculus")

  Defaults to analysis' organism.

- **ensembl_version** (str, optional) – Ensembl version of annotation to use (e.g. "grch38", "grcm38")

  Defaults to analysis' genome.

Variables **matrix_raw** (pandas.DataFrame) – DataFrame with gene expression.

plot_expression_characteristics(*matrix_raw=None*, *matrix_norm=None*, *samples=None*, *output_dir='{results_dir}/quality_control'*, *output_prefix='quality_control'*)

Plot general characteristics of the gene expression distributions within and across samples.

matrix_raw [{str, pandas.DataFrame}, optional] Name of analysis attribute with raw expression values or pandas dataframe.

Defaults to analysis' *matrix_raw*.

matrix_norm [{str, pandas.DataFrame}, optional] Name of analysis attribute with normalized expression values or pandas dataframe.

Defaults to analysis' *matrix_norm*.

samples [list, optional] List of samples to include.

Defaults to all samples in analysis

output_dir [str, optional] Directory for output files.

Defaults to "{results_dir}/quality_control"

output_prefix [str, optional] Prefix for output files.

Defaults to "quality_control"

ngs_toolkit.rnaseq.**knockout_plot**(*analysis=None*, *knockout_genes=None*, *matrix='matrix_norm'*, *samples=None*, *comparison_results=None*, *output_dir=None*, *output_prefix='knockout_expression'*, *square=True*, *rasterized=True*)

Plot expression of knocked-out genes in all samples.

Parameters

- **analysis** (*RNASeqAnalysis*, optional) – Analysis object.

  Not required if *matrix* is given.

- **knockout_genes** (`list`, optional) – List of perturbed genes to plot.

  Defaults to the set of *knockout* attributes in the analysis' samples if *analysis* is given. Otherwise must be given.

- **matrix** (*str, optional*) – Matrix with expression values to use.

  Defaults to "matrix_norm"

- **samples** (*[type], optional*) – [description]

  Defaults to `None`.

- **comparison_results** (*[type], optional*) – [description]

  Defaults to `None`.

- **output_dir** (*[type], optional*) – [description]

  Defaults to `None`.

- **output_prefix** (*str, optional*) – Prefix for output files.

  Defaults to "knockout_expression"

- **square** (*bool, optional*) – Whether heatmap cells should have inforced aspect.

  Defaults to `True`.

- **rasterized** (*bool, optional*) – Whether heatmap cells should be rasterized.

  Defaults to `True`.

ngs_toolkit.rnaseq.**assess_cell_cycle**(*analysis*, *matrix=None*, *output_dir=None*, *output_prefix='cell_cycle_assessment'*)

Predict cell cycle phase from expression data.

1.10.6 ngs_toolkit.demo

A module dedicated to the generation of Analysis, Projects and their data.

ngs_toolkit.demo.data_generator.**generate_count_matrix**(*n_factors=1*, *n_replicates=4*, *n_features=1000*, *intercept_mean=4*, *intercept_std=2*, *coefficient_stds=0.4*, *size_factors=None*, *size_factors_std=0.1*, *dispersion_function=None*)

Generate count matrix for groups of samples by sampling from a negative binomial distribution.

ngs_toolkit.demo.data_generator.**generate_data**(*n_factors=1*, *n_replicates=4*, *n_features=1000*, *coefficient_stds=0.4*, *data_type='ATAC-seq'*, *genome_assembly='hg38'*, ***kwargs*)

Creates real-looking data

Parameters

- **n_factors** (`int`, optional) – Number of factors influencing variance between groups. For each factor there will be two groups of samples.

  Defaults to 1.

- **n_replicates** (int, optional) – Number of replicates per group.

  Defaults to 4.

- **n_features** (int, optional) – Number of features (i.e. genes, regions) in matrix.

  Defaults to 1000.

- **coefficient_stds** ({int, list}, optional) – Standard deviation of the coefficients between groups. If a list, must match the number of n_factors.

  Defaults to 1.

- **data_type** (bool, optional) – Data type of the project. Must be one of the ngs_toolkit classes.

  Default is "ATAC-seq"

- **genome_assembly** (bool, optional) – Genome assembly of the project.

  Default is "hg38"

- ****kwargs** (dict) – Additional keyword arguments will be passed to *ngs_toolkit.demo.data_generator.generate_count_matrix()*.

Returns A tuple of pandas.DataFrame objects with numeric and categorical data respectively.

Return type tuple

ngs_toolkit.demo.data_generator.**generate_project**(*output_dir=None*, *project_name='test_project'*, *organism='human'*, *genome_assembly='hg38'*, *data_type='ATAC-seq'*, *n_factors=1*, *only_metadata=False*, *sample_input_files=False*, *initialize=True*, ***kwargs*)

Creates a real-looking PEP-based project with respective input files and quantification matrix.

Parameters

- **output_dir** (str, optional) – Directory to write files to.

  Defaults to a temporary location in the user's ${TMPDIR}.

- **project_name** (bool, optional) – Name for the project.

  Default is "test_project".

- **organism** (bool, optional) – Organism of the project.

  Default is "human"

- **genome_assembly** (bool, optional) – Genome assembly of the project.

  Default is "hg38"

- **data_type** (bool, optional) – Data type of the project. Must be one of the ngs_toolkit classes.

  Default is "ATAC-seq"

- **only_metadata** (obj:*bool*, optional) – Whether to only generate metadata for the project or input files in addition.

  Default is False.

- **sample_input_files** (obj:*bool*, optional) – Whether the input files for the respective data type should be produced.

  This would be BAM and peak files for ATAC-seq or BAM files for RNA-seq.

  Default is `True`.

- **initialize** (obj:*bool*, optional) – Whether the project should be initialized into an Analysis object for the respective `data_type` or simply return the path to a PEP configuration file.

  Default is `True`.

- ****kwargs** (`dict`) – Additional keyword arguments will be passed to *ngs_toolkit.demo.data_generator.generate_data()*.

Returns The Analysis object for the project or a path to its PEP configuration file.

Return type {*ngs_toolkit.analysis.Analysis*, `str`}

ngs_toolkit.demo.data_generator.**generate_projects**(*output_path=None, project_prefix_name='demo-project', data_types=['ATAC-seq', 'RNA-seq'], organisms=['human', 'mouse'], genome_assemblies=['hg38', 'mm10'], n_factors=[1, 2, 5], n_features=[100, 1000, 10000], n_replicates=[1, 3, 5], **kwargs*)

Create a list of Projects given ranges of parameters, which will be passed to *ngs_toolkit.demo.data_generator.generate_project()*.

ngs_toolkit.demo.data_generator.**generate_bam_file**(*count_vector*, *output_bam*, *genome_assembly='hg38'*, *chrom_sizes_file=None*, *index=True*)

Generate BAM file containing reads matching the counts in a vector of features

ngs_toolkit.demo.data_generator.**generate_peak_file**(*peak_set*, *output_peak*, *genome_assembly='hg38'*, *summits=False*)

Generate peak files containing regions from a fraction of a given set of features

ngs_toolkit.demo.data_generator.**generate_sample_input_files**(*analysis*, *matrix*)

Generate input files (BAM, peaks) for a sample depending on its data type.

ngs_toolkit.demo.data_generator.**initialize_analysis_of_data_type**(*data_type*, *pep_config*, **args*, ***kwargs*)

Initialize an Analysis object from a PEP config with the appropriate `data_type`.

ngs_toolkit.demo.data_generator.**get_random_genomic_locations**(*n_regions*, *width_mean=500*, *width_std=400*, *min_width=300*, *genome_assembly='hg38'*)

Get *n_regions'* number of random genomic locations respecting the boundaries of the `genome_assembly`

ngs_toolkit.demo.data_generator.**get_random_genes**(*n_genes*, *genome_assembly='hg38'*)

Get `n_genes` number of random genes from the set of genes of the `genome_assembly`

ngs_toolkit.demo.data_generator.**get_genomic_bins**(*n_bins*, *distribution='normal'*, *genome_assembly='hg38'*)

Get a `size` number of random genomic bins respecting the boundaries of the `genome_assembly`

1.10.7 ngs_toolkit.general

ngs_toolkit.general.**get_genome_reference**(*organism*, *genome_assembly=None*, *output_dir=None*, *genome_provider='UCSC'*, *file_format='2bit'*, *dry_run=False*, *overwrite=True*)

Get genome FASTA/2bit file. Saves results to disk and returns path to file.

Parameters

- **organism** (`str`) – Organism to get annotation for. Currently supported: "human" and "mouse".
- **output_dir** (`str`, optional) – Directory to write output to. Defaults to current directory
- **genome_provider** (`str`, optional) – Which genome provider to use. One of 'UCSC' or 'Ensembl'.
- **file_format** (`str`, optional) – File format to get. One of 'fasta' or '2bit'.
- **dry_run** (`bool`, optional) – Whether to not download and just return path to file.
- **overwrite** (`bool`, optional) – Whether existing files should be overwritten by new ones. Otherwise they will be kept and no action is made. Defaults to True.

Returns Path to genome FASTA/2bit file, but if *dry_run* tuple of URL of reference genome and path to file.

Return type {str, tuple}

Raises **ValueError** – If arguments are not in possible options or if desired combination is not available.

ngs_toolkit.general.**get_blacklist_annotations**(*organism*, *genome_assembly=None*, *output_dir=None*, *overwrite=True*)

Get annotations of blacklisted genomic regions for a given organism/genome assembly. Saves results to disk and returns a path to a BED file.

Parameters

- **organism** (`str`) – Organism to get annotation for. Currently supported: "human" and "mouse".
- **genome_assembly** (`str`, optional) – Ensembl assembly/version to use. Default for "human" is "hg19/grch37" and for "mouse" is "mm10/grcm38".
- **output_dir** (`str`, optional) – Directory to write output to. Defaults to "reference" in current directory.
- **overwrite** (`bool`, optional) – Whether existing files should be overwritten by new ones. Otherwise they will be kept and no action is made. Defaults to True.

Returns Path to blacklist BED file

Return type str

ngs_toolkit.general.**get_tss_annotations**(*organism, genome_assembly=None, save=True, output_dir=None, chr_prefix=True, gene_types=['protein_coding', 'processed_transcript', 'lincRNA', 'antisense'], overwrite=True*)

Get annotations of TSS for a given organism/genome assembly. This is a simple approach using Biomart's API querying the Ensembl database. Saves results to disk and returns a dataframe.

Parameters

- **organism** (str) – Organism to get annotation for. Currently supported: "human" and "mouse".
- **genome_assembly** (str, optional) – Ensembl assembly/version to use. Default for "human" is "grch37" and for "mouse" is "grcm38".
- **save** (bool, optional) – Whether to save to disk under output_dir. Defaults to True.
- **output_dir** (str, optional) – Directory to write output to. Defaults to "reference" in current directory.
- **chr_prefix** (bool, optional) – Whether chromosome names should have the "chr" prefix. Defaults to True
- **gene_types** (list, optional) – Subset of transcript biotypes to keep. See here the available biotypes https://www.ensembl.org/Help/Faq?id=468 Defaults to 'protein_coding', 'processed_transcript', 'lincRNA', 'antisense'.
- **overwrite** (bool, optional) – Whether existing files should be overwritten by new ones. Otherwise they will be kept and no action is made. Defaults to True.

Returns DataFrame with genome annotations

Return type pandas.DataFrame

ngs_toolkit.general.**get_genomic_context**(*organism, genome_assembly=None, save=True, output_dir=None, chr_prefix=True, region_subset=['promoter', 'exon', '5utr', '3utr', 'intron', 'genebody', 'intergenic'], gene_types=['protein_coding', 'processed_transcript', 'lincRNA', 'antisense'], promoter_width=3000, overwrite=True*)

Get annotations of TSS for a given organism/genome assembly. This is a simple approach using Biomart's API querying the Ensembl database. Saves results to disk and returns a dataframe.

The API call to BioMart can take a bit, so the function should take ~4 min for a human genome.

Parameters

- **organism** (str) – Organism to get annotation for. Currently supported: "human" and "mouse".
- **genome_assembly** (str, optional) – Ensembl assembly/version to use. Default for "human" is "grch37" and for "mouse" is "grcm38".
- **save** (bool, optional) – Whether to save to disk under output_dir. Defaults to True.
- **output_dir** (str, optional) – Directory to write output to. Defaults to "reference" in current directory.
- **chr_prefix** (bool, optional) – Whether chromosome names should have the "chr" prefix. Defaults to True
- **gene_types** (list, optional) – Subset of transcript biotypes to keep. See here the available biotypes https://www.ensembl.org/Help/Faq?id=468 Defaults to 'protein_coding', 'processed_transcript', 'lincRNA', 'antisense'.

- **overwrite** (bool, optional) – Whether existing files should be overwritten by new ones. Otherwise they will be kept and no action is made. Defaults to True.

Returns DataFrame with genome annotations

Return type pandas.DataFrame

ngs_toolkit.general.**deseq_analysis**(*count_matrix*, *experiment_matrix*, *comparison_table*, *formula*, *output_dir*, *output_prefix*, *overwrite=True*, *alpha=0.05*, *independent_filtering=False*, *create_subdirectories=True*, *save_inputs=True*, ***kwargs*)

Perform differential comparison analysis with DESeq2.

Note: Do not include hyphens ("-") in any of the samples or groups names! R freaks out with this.

TODO: fix hyphens in names issue

Parameters

- **count_matrix** (pandas.DataFrame) – Data frame of shape (samples, variables) with raw read counts.
- **experiment_matrix** (pandas.DataFrame) – Data frame with columns "sample_name" and any other variables used in the *formula*.
- **comparison_table** (pandas.DataFrame) – Data frame with columns "comparison_name", "sample_group" and sample_name".
- **formula** (str) – Formula to test in R/patsy notation. Usually something like "~ batch + group".
- **output_dir** (str) – Output directory for produced files.
- **output_prefix** (str) – Prefix to add to produced files.
- **overwrite** (bool, optional) – Whether files existing should be overwritten. Defaults to True.
- **alpha** (*number, optional*) – Significance level to reject null hypothesis. This in practice has no effect as results for all features will be returned. Defaults to 0.05.
- **create_subdirectories** (bool) – Whether to create subdirectories for the result of each comparison.
- ****kwargs** (dict) – Additional keyword arguments to be passed to the DESeq function of DESeq2.

Returns Data frame with results, statistics for each feature.

Return type pandas.DataFrame

ngs_toolkit.general.**least_squares_fit**(*matrix*, *design_matrix*, *test_model*, *null_model='~ 1'*, *standardize_data=True*, *multiple_correction_method='fdr_bh'*)

Fit a least squares model with only categorical predictors. Computes p-values by comparing the log likelihood ratio of the chosen model to a *null_model*.

Parameters

- **matrix** (pandas.DataFrame) – A Data frame of shape (samples, variables).
- **design_matrix** (pandas.DataFrame) – A Data frame of shape (samples, variables) with all the variables in *test_model*.
- **test_model** (str) – Model design to test in R/patsy notation.

- **null_model** (str, optional) – Null model design in R/patsy notation. Defaults to "~ 1".
- **standardize_data** (bool, optional) – Whether data should be standardized prior to fitting. Defaults to True.
- **multiple_correction_method** (str, optional) – Method to use for multiple test correction. See statsmodels.sandbox.stats.multicomp.multipletests. Defaults to "fdr_bh".

Returns

- *pandas.DataFrame* – Statistics of model fitting and comparison between models for each feature.
- *Example:*
- *matrix = np.random.random(10000000).reshape(100, 100000)*
- *P = np.concatenate([[0] * 50, [1] * 50]) # dependent variable*
- *Q = np.concatenate([[0] * 25, [1] * 25] + [[0] * 25, [1] * 25]) # covariate*
- *design_matrix = pd.DataFrame([P, Q], index=["P", "Q"]).T*
- *matrix = matrix.T * ((1 + design_matrix.sum(axis=1)) * 4).values*
- *matrix = pd.DataFrame(matrix.T)*
- *test_model = "~ Q + P"*
- *null_model = "~ Q"*
- *res = least_squares_fit(matrix, design_matrix, test_model, null_model)*
- *res.head()*

ngs_toolkit.general.**differential_from_bivariate_fit**(*comparison_table*, *matrix*, *output_dir*, *output_prefix*, *n_bins=250*, *multiple_correction_method='fdr_bh'*, *plot=True*, *palette='colorblind'*, *make_values_positive=False*)

Perform differential analysis using a bivariate gaussian fit on the relationship between mean and fold-change for each comparison.

Parameters

- **comparison_table** (pandas.DataFrame) – Dataframe with 'comparison_name', 'comparison_side' and 'sample_name', 'sample_group' columns.
- **matrix** (pandas.DataFrame) – Matrix of *n_features, n_samples* with normalized, log-transformed values to perform analysis on.
- **output_dir** (str) – Output directory
- **output_prefix** (str) – Prefix for outputs.
- **n_bins** (int) – Number of bins of mean values along which to standardize fold-changes.
- **multiple_correction_method** (str) – Multiple correction method from *statsmodels.sandbox.stats.multicomp.multipletests*.
- **plot** (bool) – Whether to generate plots.
- **palette** (str) – Color palette to use. This can be any matplotlib palette and is passed to *sns.color_palette*.

- **make_values_positive** (bool) – Whether to transform *matrix* to have minimum value 0. Default False.

Returns Results of fitting and comparison between groups for each feature.

Return type pandas.DataFrame

ngs_toolkit.general.**lola**(*bed_files*, *universe_file*, *output_folder*, *genome*, *output_prefixes=None*, *cpus=8*)

Perform location overlap analysis (LOLA).

If bed_files is a list with more than one element, use `output_prefixes` to pass a list of prefixes to label the output files for each input BED file.

Files will be created in `output_folder` mimicking the output that the R function LOLA::writeCombinedEnrichment writes.

Requires the R package "LOLA" to be installed:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("LOLA")
```

Parameters

- **bed_files** (*str,list*) – A string path to a BED file or a list of paths.
- **universe_file** (str) – A path to a BED file representing the universe from where the BED file(s) come from.
- **output_folder** (str) – Output folder for resulting files.
- **genome** (str, optional) – Genome assembly from which the BED files come from. This is used to get the LOLA databases from the ngs_toolkit._CONFIG parameters.
- **output_prefixes** (list, optional) – A list of strings with prefixes to be used in case `bed_files` is a list.
- **cpus** (int, optional) – Number of CPUs/threads to use. Defaults to 8

ngs_toolkit.general.**homer_combine_motifs**(*comparison_dirs*, *output_dir*, *region_prefix='differential_analysis'*, *reduce_threshold=0.6*, *match_threshold=10*, *info_value=0.6*, *p_value_threshold=1e-25*, *fold_enrichment=None*, *cpus=8*, *run=True*, *distributed=True*, *genome='hg19'*, *motif_database=None*)

Create consensus of de novo discovered motifs from HOMER

Parameters

- **comparison_dirs** (list) – Iterable of comparison directories where homer was run. Should contain a "homerMotifs.all.motifs" file.
- **output_dir** (str) – Output directory.
- **p_value_threshold** (*number, optional*) – Threshold for inclusion of a motif in the consensus set. Defaults to 1e-5
- **cpus** (*number, optional*) – Number of available CPUS/threads for multithread processing. Defaults to 8
- **run** (bool, optional) – Whether to run enrichment of each comparison in the consensus motifs. Default is True

- **distributed** (bool, optional) – Whether to run enrichment as a cluster job. Default is True
- **genome** (str) – Genome assembly of the data. Default is 'hg19'.
- **motif_database** (str) – Motif database to restrict motif matching too.

Returns If *run* is *False*, returns path to consensus motif file. Otherwise *None*.

Return type {str,None}

ngs_toolkit.general.**enrichr**(*dataframe*, *gene_set_libraries=None*, *kind='genes'*, *max_attempts=5*)

Use Enrichr on a list of genes (currently only genes supported through the API).

Parameters

- **dataframe** (str) – DataFrame with column "gene_name".
- **gene_set_libraries** (list, optional) – Gene set libraries to use. Defaults to values in initial configuration file. To see them, do: `ngs_toolkit._CONFIG['resources']['enrichr']['gene_set_libraries']`
- **kind** (str, optional) – Type of input. Right now, only "genes" is supported. Defaults to "genes"
- **max_attempts** (int, optional) – Number of times to try a call to Enrichr API. Defaults to 5

Returns Results of enrichment analysis

Return type pandas.DataFrame

Raises **Exception** – If *max_attempts* is exceeded

ngs_toolkit.general.**run_enrichment_jobs**(*results_dir, genome, background_bed, steps=['lola', 'meme', 'homer', 'enrichr'], overwrite=True, pep_config=None*)

Submit parallel enrichment jobs for a specific analysis.

Parameters

- *param results_dir:* – Directory with files prepared by ngs_toolkit.general.run_enrichment_jobs
- *param genome:* – Genome assembly of the analysis.
- **background_bed** (str) – BED file to use as background for LOLA analysis. Typically the analysis' own consensus region set.
- **steps** (list, optional) – Steps of the analysis to perform. Defaults to ["region", lola", "meme", "homer", "enrichr"].
- *param overwrite: bool, optional* – Whether output should be overwritten. In this case no jobs will be submitted for jobs with existing output files. Defaults to True
- param pep_config: str, optional – Pickle file of the analysis. Only required for "region" enrichment.

ngs_toolkit.general.**project_to_geo**(*project*, *output_dir='geo_submission'*, *samples=None*, *distributed=False*, *dry_run=False*, ***kwargs*)

Prepare raw sequencing files for submission to GEO. Files will be copied or generated in a new directory `output_dir`. It will get the raw BAM file(s) of each sample, and in case of ATAC-seq/ChIP-seq samples, the bigWig and peak files. If multiple BAM files exist for each sample, all will be copied and sequencially named with the "fileN" suffix, where "N" is the file number.

For each copied file a md5sum will be calculated.

A pandas DataFrame with info on the sample's files and md5sums will be returned.

Variables

- **project** (peppy.Project) – A peppy.Project object to process.
- **output_dir** (str, optional) – Directory to create output. Will be created/overwriten if existing.

  Defaults to "geo_submission".
- **samples** (list, optional) – List of peppy.Sample objects in project to restrict to.

  Defaults to all samples in project.
- **distributed** (bool, optional) – Whether processing should be distributed as jobs in a computing cluster for each sample. Currently available implementation supports a SLURM cluster only.

  Defaults is False.
- **dry_run** (bool, optional) – Whether copy/execution/submisison commands should be not be run, to test.

  Default is False.
- ****kwargs** (dict) – Additional keyword arguments will be passed to *ngs_toolkit.utils.submit_job()* if distributed is True, and on to a divvy submission template. Pass for example:
  - computing_configuration="slurm"
  - jobname="job"
  - cores=2
  - mem=8000
  - partition="longq"

Returns Annotation of samples and their BAM, BigWig, narrowPeak files and respective md5sums.

Return type pandas.DataFrame

ngs_toolkit.general.**rename_sample_files**(*annotation_mapping*, *old_sample_name_column='old_sample_name'*, *new_sample_name_column='new_sample_name'*, *tmp_prefix='rename_sample_files'*, *results_dir='results_pipeline'*, *dry_run=False*)

Rename existing directories with pipeline outputs for samples based on mapping of old/new sample names.

All files within the directory with the old sample name will be renamed recursively. Old and new sample names can overlap - this procedure will handle these cases correctly by a 2-step process with temporary sample names with prefix tmp_prefix.

> **Caution:** NEEDS TESTING! This function has not been used in a while and needs more testing.

Variables

- **annotation_mapping** (pandas.DataFrame) – DataFrame with mapping of old (column "previous_sample_name") vs new ("new_sample_name") sample names.

- **old_sample_name_column** (str, optional) – Name of column with old sample names.

  Defaults to "old_sample_name".

- **new_sample_name_column** (str, optional) – Name of column with new sample names.

  Defaults to "new_sample_name".

- **tmp_prefix** (str, optional) – Prefix for temporary files to avoid overlap between old and new names.

  Defaults to "rename_sample_files".

- **results_dir** (str, optional) – Pipeline output directory containing sample output directories.

  Defaults to "results_pipeline".

- **dry_run** (bool, optional) – Whether to print commands instead of running them.

  Defaults to False.

ngs_toolkit.general.**query_biomart**(*attributes=None*, *species='hsapiens'*, *ensembl_version='grch37'*, *max_api_retries=5*)

Query Biomart (https://www.ensembl.org/biomart/martview/).

Query Biomart for gene attributes. Returns pandas dataframe with query results. If a certain field contains commas, it will attemp to return dataframe but it might fail.

Parameters

- **attributes** (list, optional) – List of ensembl atrributes to query.

  Defaults to ["ensembl_gene_id", "external_gene_name", "hgnc_id", "hgnc_symbol"].

- **species** (str, optional) – Ensembl string of species to query. Must be vertebrate.

  Defaults to "hsapiens".

- **ensembl_version** (str, optional) – Ensembl version to query. Currently "grch37", "grch38" and "grcm38" are tested.

  Defaults to "grch37".

- **max_api_retries** (int, optional) – How many times to try .

  Defaults to "grch37".

Returns Dataframe with required attributes for each entry.

Return type pandas.DataFrame

ngs_toolkit.general.**subtract_principal_component**(*x*, *pc=1*, *norm=False*, *plot=True*, *plot_name='PCA_based_batch_correction.svg'*, *max_pcs_to_plot=6*)

Given a matrix (n_samples, n_variables), remove *pc* (1-based) from matrix.

ngs_toolkit.general.**subtract_principal_component_by_attribute**(*df*, *attributes*, *pc=1*)

Given a matrix (n_samples, n_variables), remove *pc* (1-based) from matrix.

ngs_toolkit.general.**fix_batch_effect_limma**(*matrix*, *batch_variable='batch'*, *covariates=None*)

Fix batch effect in matrix using limma.

Requires the R package "limma" to be installed:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("limma")
```

Parameters

- **matrix** (pandas.DataFrame) – DataFrame with MultiIndex for potential covariate annotations
- **formula** (str, optional) – Model formula to regress out Defaults to "~batch"

Returns Regressed out matrix

Return type pandas.DataFrame

1.10.8 ngs_toolkit.graphics

ngs_toolkit.graphics.**barmap**(*x*, *figsize=None*, *square=False*, *row_colors=None*, *z_score=None*, *ylims=None*)

Plot a heatmap-style grid with barplots.

Parameters

- **x** (pandas.DataFrame) – DataFrame with numerical values to plot. If DataFrame, indexes will be used as labels.
- **figsize** (tuple) – Size in inches (width, height) of figure to produce.
- **square** (bool) – Whether resulting figure should be square.
- **row_colors** (list) – Iterable of colors to use for each row.
- **z_score** (int) – Whether input matrix *x* should be Z-score transformed row-wise (0) or column-wise (1).

Returns Figure object

Return type matplotlib.pyplot.Figure

Raises **ValueError** – If length of row_colors does not match size of provided Y axis from matrix x.

ngs_toolkit.graphics.**radar_plot**(*data*, *subplot_var='patient_id'*, *group_var='timepoint'*, *radial_vars=['NaiveBcell', 'SwitchedBcell', 'UnswitchedBcell']*, *cmap='inferno'*, *scale_to_max=True*)

Heavy inspiration from here: https://matplotlib.org/examples/api/radar_chart.html

Parameters

- **data** (pandas.DataFrame)
- **subplot_var** (str)
- **group_var** (str)
- **radial_vars** (list)
- **cmap** (str) – Matplotlib colormap to use.
- **scale_to_max** (bool) – Whether values will be scaled

ngs_toolkit.graphics.**plot_projection**(*df*, *color_dataframe*, *dims*, *output_file*, *attributes_to_plot*, *plot_max_dims=8*, *rasterized=False*, *plot_group_centroids=True*, *axis_ticklabels=True*, *axis_ticklabels_name='PC'*, *axis_lines=True*, *legends=False*, *always_legend=False*)

Plot a low dimentionality projection of samples.

Parameters

- **df** (pandas.DataFrame) – Dataframe with sample projections.
- **color_dataframe** (pandas.DataFrame) – Dataframe of RGB tuples for sample i in attribute j.
- **dims** (int) – Number of dimentions to plot
- **output_file** (str) – Path to figure output file
- **attributes_to_plot** (list) – List of levels in df.index to plot
- **plot_max_dims** (*number, optional*) – Maximum number of dimentions to plot. Defaults to 8.
- **plot_group_centroids** (bool, optional) – Whether centroids of each sample group should be plotted alongside samples. Will be square shaped. Defaults to True.
- **axis_ticklabels** (bool, optional) – Whether axis ticks and tick labels should be plotted. Defaults to False.
- **axis_lines** (bool, optional) – Whether (0, 0) dashed lines should be plotted. Defaults to True.
- **legends** (bool, optional) – Whether legends for group colours should be plotted. Defaults to False.
- **always_legend** (bool, optional) – Whether legends for group colours should be plotted in every figure panel. If False, will plot just on first/last figure panel. Defaults to False.

ngs_toolkit.graphics.**plot_region_context_enrichment**(*enr*, *output_dir='results'*, *output_prefix='region_type_enrichment'*, *across_attribute=None*, *pvalue=0.05*, *top_n=5*)

Plot results of ATACSeqAnalysis.region_context_enrichment.

Parameters

- **enr** (pandas.DataFrame) – Results of region_context_enrichment.
- **output_dir** (str, optional) – Directory to save plots to. Defaults to "results".
- **optional output_prefix** (str, optional) – Prefix to use when saveing plots. Defaults to "region_type_enrichment"
- **across_attribute** (str, optional) – Column in enrichment matrix to plot results across e.g. "comparison_name" when results matrix contains the result of various comparisons. Defaults to None (not used).
- **pvalue** (*float, optional*) – Value at which to plot a line marking the significant level. Defaults to 0.05.
- **top_n** (int, optional) – Number of features to label in volcano plot. Defaults to 5.

ngs_toolkit.graphics.**plot_comparison_correlations**(*diff*, *output_dir*, *output_prefix='comparison_correlations'*)

Plot pairwise log fold changes for various comparisons.

Parameters

- **diff** (pandas.DataFrame) – Dataframe with differential results
- **output_dir** (str) – Output directory for plots.
- **output_prefix** (str, optional) – Prefix for plots. Defaults to "comparison_correlations"

1.10.9 ngs_toolkit.utils

ngs_toolkit.utils.**have_unbuffered_output**()
Set unbuffered output for current session.

ngs_toolkit.utils.**is_running_inside_ipython**()
Check whether code is running inside an IPython session.

ngs_toolkit.utils.**get_timestamp**(*fmt='%Y-%m-%d-%H:%M:%S'*)
Get current timestamp in fmt format.

ngs_toolkit.utils.**remove_timestamp_if_existing**(*file*)
Remove timestamp from path if matching timestamp pattern exists.

ngs_toolkit.utils.**get_this_file_or_timestamped**(*file*, *permissive=True*)
Get a path to an existing timestamped file based on an non-timestamped path.

Parameters

- **file_name** (str) – File name of analysis output to record.
- **permissive** (bool) – Whether failure to find timestamped file should return the original file or raise a IndexError.

Raises **IndexError** – If not *permissive* and can't find timestamped file.

ngs_toolkit.utils.**is_analysis_descendent**(*exclude_functions=None*)
Check whether any call in the traceback comes from a function part of a ngs_toolkit.Analysis() object.

Parameters **exclude_functions** (list) – List of function names to exclude from.

Returns If is descentent, returns tuple of (Analysis instance, function name), othewise returns False.

Return type tuple

ngs_toolkit.utils.**record_analysis_output**(*file_name*, ***kwargs*)
Register a file that is an output of an Analysis. The file will be associated with the function that produced it and saved in the attribute output_files.

Parameters

- **file_name** (str) – File name of analysis output to record.
- ****kwargs** (bool) – Keyword arguments passed to *ngs_toolkit.utils.is_analysis_descendent()*.

ngs_toolkit.utils.**submit_job**(*code*, *job_file*, *log_file=None*, *computing_configuration=None*, *dry_run=False*, *limited_number=False*, *total_job_lim=500*, *refresh_time=10*, *in_between_time=5*, ***kwargs*)
Submit a job to be run. Uses divvy to allow running on a local computer or distributed computing resources.

Parameters

- **code** (str) – String of command(s) to be run.
- **job_file** (str) – File to write job code to.

- **log_file** (str) – Log file to write job output to.

  Defaults to job_file with ".log" ending.

- **computing_configuration** (str) – Name of divvy computing configuration to use.

  Defaults to 'default' which is to run job in localhost.

- **dry_run** (bool) – Whether not to actually run job.

  Defaults to False.

- **limited_number** (bool) – Whether to restrict jobs to a maximum number. Currently only possible if using "slurm".

  Defaults to False.

- **total_job_lim** (int) – Maximum number of jobs to restrict to.

  Defaults to 500.

- **refresh_time** (int) – Time in between checking number of jobs in seconds.

  Defaults to 10.

- **in_between_time** (int) – Time in between job submission in seconds.

  Defaults to 5.

- ****kwargs** (dict) – Additional keyword arguments will be passed to the chosen submission template according to *computing_configuration*. Pass for example: jobname="job", cores=2, mem=8000, partition="longq".

ngs_toolkit.utils.**chunks**(*l*, *n*)
Partition iterable *l* in chunks of size *n*.

Parameters

- **l** (*iterable*) – Iterable (e.g. list or numpy array).
- **n** (int) – Size of chunks to generate.

ngs_toolkit.utils.**sorted_nicely**(*l*)
Sort an iterable in the way that humans expect.

Parameters **l** (*iterable*) – Iterable to be sorted

Returns Sorted iterable

Return type iterable

ngs_toolkit.utils.**standard_score**(*x*)
Compute a standard score, defined as (x - min(x)) / (max(x) - min(x)).

Parameters **x** (numpy.ndarray) – Numeric array.

Returns Transformed array.

Return type numpy.ndarray

ngs_toolkit.utils.**z_score**(*x*)
Compute a Z-score, defined as (x - mean(x)) / std(x).

Parameters **x** (numpy.ndarray) – Numeric array.

Returns Transformed array.

Return type numpy.ndarray

ngs_toolkit.utils.**logit**(*x*)
Compute the logit of x, defined as log(x / (1 - x)).

Parameters **x** (numpy.ndarray) – Numeric array.

Returns Transformed array.

Return type numpy.ndarray

ngs_toolkit.utils.**count_dataframe_values**(*x*)
Count number of non-null values in a dataframe.

Parameters **x** (pandas.DataFrame) – Pandas DataFrame

Returns Number of non-null values.

Return type int

ngs_toolkit.utils.**location_index_to_bed**(*index*)
Get a pandas DataFrame with columns "chrom", "start", "end" from an pandas Index of strings in form "chrom:start-end".

Parameters **index** ({list, pandas.Index, pandas.Series), pandas.DataFrame)}) – Index strings of the form "chrom:start-end".

Returns Pandas dataframe.

Return type pandas.DataFrame

ngs_toolkit.utils.**bed_to_index**(*df*)
Get an index of the form chrom:start-end from a a dataframe with such columns.

Parameters **df** ({pandas.DataFrame, pybedtools.bedtool.BedTool, str}) – DataFrame with columns "chrom", "start" and "end".

Returns Pandas index.

Return type pandas.Index

ngs_toolkit.utils.**bedtool_to_index**(*bedtool*)
Convert bedtool or path to a bed file to list of region identifiers

ngs_toolkit.utils.**to_bed_index**(*sites*)
Convert bedtool, BED file or dataframe to list of region identifiers

ngs_toolkit.utils.**sort_bed_nicely**(*bed_file*)
Sorts BED file but in sorted_nicely order

ngs_toolkit.utils.**timedelta_to_years**(*x*)
Convert a timedelta to years.

Parameters **x** (pandas.Timedelta) – A Timedelta object.

Returns Years.

Return type float

ngs_toolkit.utils.**signed_max**(*x*, *f=0.66*, *axis=0*)
Return maximum or minimum of array *x* depending on the sign of the majority of values. If there isn't a clear majority (at least *f* fraction in one side), return mean of values. If given a pandas DataFrame or 2D numpy array, will apply this across rows (columns-wise, axis=0) or across columns (row-wise, axis=1). Will return NaN for non-numeric values.

Parameters

- **x** ({numpy.ndarray, pandas.DataFrame, pandas.Series}) – Input values to reduce
- **f** (float) – Threshold fraction of majority agreement.

  Default is 0.66.

- **axis** (int) – Whether to apply across rows (0, column-wise) or across columns (1, row-wise).

  Default is 0.

Returns Pandas Series with values reduced to the signed maximum.

Return type pandas.Series

ngs_toolkit.utils.**log_pvalues**(*x*, *f=0.1*)
Calculate -log10(p-value) of array.

Replaces infinite values with:

```
max(x) + max(x) * f
```

that is, fraction f more than the maximum non-infinite -log10(p-value).

Parameters

- **x** (pandas.Series) – Series with numeric values
- **f** (float) – Fraction to augment the maximum value by if x contains infinite values.

  Defaults to 0.1.

Returns Transformed values.

Return type pandas.Series

ngs_toolkit.utils.**r2pandas_df**(*r_df*)
Make pandas.DataFrame from a R dataframe given by rpy.

ngs_toolkit.utils.**recarray2pandas_df**(*recarray*)
Make pandas.DataFrame from numpy.recarray.

ngs_toolkit.utils.**collect_md5_sums**(*df*)
Given a dataframe with columns with paths to md5sum files ending in '_md5sum', replace the paths to the md5sum files with the actual checksums.

Useful to use in combination with *project_to_geo()*.

Parameters **df** (pandas.DataFrame) – A dataframe with columns ending in '_md5sum'.

Returns DataFrame with md5sum columns replaced with the actual md5sums.

Return type pandas.DataFrame

ngs_toolkit.utils.**decompress_file**(*file*, *output_file=None*)
Decompress a gzip-compressed file out-of-memory. Output default is same as file without ".gz" ending.

ngs_toolkit.utils.**compress_file**(*file*, *output_file=None*)
Compress a gzip-compressed file out-of-memory. Output default is same as file but with ".gz" ending.

ngs_toolkit.utils.**download_file**(*url*, *output_file*, *chunk_size=1024*)
Download a file and write to disk in chunks (not in memory).

Parameters

- **url** (str) – URL to download from.
- **output_file** (str) – Path to file as output.
- **chunk_size** (int) – Size in bytes of chunk to write to disk at a time.

ngs_toolkit.utils.**download_gzip_file**(*url*, *output_file*, ***kwargs*)
Download a gzip compressed file and write uncompressed file to disk in chunks (not in memory).

Parameters

- **url** (str) – URL to download from.
- **output_file** (str) – Path to file as output.
- ****kwargs** (dict) – Additional keyword arguments are passed to *ngs_toolkit.utils.download_file()*.

ngs_toolkit.utils.**deseq_results_to_bed_file**(*deseq_result_file*, *bed_file*, *sort=True*, *ascending=False*, *normalize=False*, *significant_only=False*, *alpha=0.05*, *abs_fold_change=1.0*)

Write BED file with fold changes from DESeq2 as score value.

ngs_toolkit.utils.**homer_peaks_to_bed**(*homer_peaks*, *output_bed*)

Convert HOMER peak calls to BED format. The fifth column (score) is the -log10(p-value) of the peak.

Parameters

- **homer_peaks** (str) – HOMER output with peak calls.
- **output_bed** (str) – Output BED file.

ngs_toolkit.utils.**macs2_call_chipseq_peak**(*signal_samples*, *control_samples*, *output_dir*, *name*, *distributed=True*)

Call ChIP-seq peaks with MACS2 in a slurm job.

Parameters

- **signal_samples** (list) – Signal Sample objects.
- **control_samples** (list) – Background Sample objects.
- **output_dir** (list) – Parent directory where MACS2 outputs will be stored.
- **name** (str) – Name of the MACS2 comparison being performed.
- **distributed** (bool) – Whether to submit a SLURM job or to return a string with the runnable.

ngs_toolkit.utils.**homer_call_chipseq_peak_job**(*signal_samples*, *control_samples*, *output_dir*, *name*, *distributed=True*)

Call ChIP-seq peaks with MACS2 in a slurm job.

Parameters

- **signal_samples** (list) – Signal Sample objects.
- **control_samples** (list) – Background Sample objects.
- **output_dir** (list) – Parent directory where MACS2 outputs will be stored.
- **name** (str) – Name of the MACS2 comparison being performed.

ngs_toolkit.utils.**bed_to_fasta**(*input_bed*, *output_fasta*, *genome_file*)

Retrieves DNA sequence underlying specific region. Names of FASTA entries will be of form `chr:start-end`.

Parameters

- **input_bed** (str) – Path to input BED file.
- **output_fasta** (str) – Path to resulting FASTA file.
- **genome_file** (str) – Path to genome file in either 2bit or FASTA format. Will be guessed based on file ending.

Raises **ValueError** – If *genome_file* format cannot be guessed or is not supported.

ngs_toolkit.utils.**read_bed_file_three_columns**(*input_bed: str*) → pandas.core.frame.DataFrame
Read BED file into dataframe, make 'name' field from location.

ngs_toolkit.utils.**bed_to_fasta_through_2bit**(*input_bed*, *output_fasta*, *genome_2bit*)
Retrieves DNA sequence underlying specific region. Requires the *twoBitToFa* command from UCSC kent tools. Names of FASTA entries will be of form "chr:start-end".

Parameters

- **input_bed** (`str`) – Path to input BED file.
- **output_fasta** (`str`) – Path to resulting FASTA file.
- **genome_2bit** (`str`) – Path to genome 2bit file.

ngs_toolkit.utils.**bed_to_fasta_through_fasta**(*input_bed*, *output_fasta*, *genome_fasta*)
Retrieves DNA sequence underlying specific region. Uses `bedtools getfasta` (internally through pybedtools.BedTool.sequence). Names of FASTA entries will be of form "chr:start-end".

Parameters

- **input_bed** (`str`) – Path to input BED file.
- **output_fasta** (`str`) – Path to resulting FASTA file.
- **genome_fasta** (`str`) – Path to genome FASTA file.

ngs_toolkit.utils.**count_reads_in_intervals**(*bam*, *intervals*, *permissive=True*)
Count total number of reads in a iterable holding strings representing genomic intervals of the form `"chrom:start-end"`.

Please make sure both `intervals` and `bam` file are zero- or one-indexed.

Parameters

- **bam** (`str`) – Path to BAM file.
- **intervals** (`list`) – List of strings with genomic coordinates in format `"chrom:start-end"`.

Returns Dict of read counts for each interval.

Return type `dict`

ngs_toolkit.utils.**normalize_quantiles_r**(*array*)
Quantile normalization with a R implementation.

Requires the R package "preprocessCore" to be installed:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("preprocessCore")
```

Parameters **array** (`numpy.ndarray`) – Numeric array to normalize.

Returns Normalized numeric array.

Return type `numpy.ndarray`

ngs_toolkit.utils.**normalize_quantiles_p**(*df_input*)
Quantile normalization with a ure Python implementation. Code from https://github.com/ShawnLYU/Quantile_Normalize.

Parameters **df_input** (`pandas.DataFrame`) – Dataframe to normalize.

Returns Normalized numeric array.

Return type `numpy.ndarray`

ngs_toolkit.utils.**cqn**(*matrix*, *gc_content*, *lengths*)
Conditional quantile normalization (CQN) with the `cqn` R library. It uses GC content and length of regulatory elements as covariates.

Requires the R package "cqn" to be installed:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("cqn")
```

Parameters

- **matrix** (`pandas.DataFrame`) – DataFrame to normalize.
- **gc_content** (`pandas.Series`) – Series with GC content of each feature in `matrix`.
- **lengths** (`pandas.Series`) – Series with length of each feature in `matrix`.

Returns Normalized DataFrame

Return type `pandas.DataFrame`

ngs_toolkit.utils.**count_bam_file_length**(*bam_file: str*) → int
Get length of BAM indexed file

ngs_toolkit.utils.**count_lines**(*file: str*) → int
Count lines of plain text file

ngs_toolkit.utils.**get_total_region_area**(*bed_file: str*) → int
Get sum of BED records

ngs_toolkit.utils.**get_region_lengths**(*bed_file: str*) → pandas.core.series.Series
Get length of each record in BED file

ngs_toolkit.utils.**get_regions_per_chromosomes**(*bed_file: str*) → pandas.core.series.Series
Count record per chromosome in BED file

1.10.10 ngs_toolkit.parsers

ngs_toolkit.parsers.**parse_ame**(*ame_output*)
Parse results of MEME-AME motif enrichment.

Parameters **ame_output** (`str`) – MEME-AME results file.

Returns Data frame with enrichment statistics for each found TF motif.

Return type pandas.DataFrame

Raises **IOError** – If directory contain

ngs_toolkit.parsers.**parse_homer**(*homer_dir*)
Parse results of HOMER findMotifs.pl de novo motif enrichment.

Parameters **homer_dir** (`str`) – Directory with HOMER results.

Returns Data frame with enrichment statistics for each found TF motif.

Return type pandas.DataFrame

Raises **IOError** –

ngs_toolkit.parsers.**parse_great_enrichment**(*input_tsv*)
Parse output from GREAT enrichment (http://great.stanford.edu).

Parameters **input_tsv** (`str`) – TSV file exported from GREAT through the option "All data as .tsv" in "Global Controls".

Returns Pandas dataframe with enrichment results.

Return type pandas.DataFrame

1.10.11 ngs_toolkit

ngs_toolkit.**setup_logger**(*level='INFO'*, *logfile=None*)

Set up a logger for the library.

Parameters

- **level** (str, optional) – Level of logging to display. See possible levels here: https://docs.python.org/2/library/logging.html#levels

  Defaults to "INFO".

- **logfile** (str, optional) – File to write log to.

  Defaults to "~/.ngs_toolkit.log.txt".

Returns A logger called "ngs_toolkit".

Return type logging.Logger

ngs_toolkit.**setup_config**(*custom_yaml_config=None*)

Set up global library configuration.

It reads ngs_toolkit's package data to load a default configuration, tries to update it by reading a file in ~/.ngs_toolkit.config.yaml if present, and lastly, updates it by reading a possible passed yaml file custom_yaml_config. Non-exisiting fields will maintain the previous values, so that the user needs only to specify the section(s) as needed.

Parameters **custom_yaml_config** (str, optional) – Path to YAML file with configuration. To see the structure of the YAML file, see https://github.com/afrendeiro/toolkit/blob/master/ngs_toolkit/config/default.yaml

Defaults to None.

Returns Dictionary with configurations.

Return type dict

ngs_toolkit.**setup_graphic_preferences**()

Set up graphic preferences.

It uses the values under "preferences:graphics:matplotlib:rcParams" and "preferences:graphics:seaborn:parameters" to matplotlib and seaborn respectively.

1.11 Testing

To make sure everything is correctly configured, the user is encouraged to test the library prior to use.

In order to do this, install testing requirements and simply run pytest:

```
pip install ngs-toolkit[testing]
pytest --pyargs ngs_toolkit
```

Pytest will output summary results (see for example) and further outputs can be seen in ${TMPDIR}/pytest-of-${USER}/ or /tmp/pytest-of-${USER}/ if $TMPDIR is not defined.

1.12 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

1.12.1 [Unreleased]

Added

- New CRISPR module for the analysis of pooled CRISPR screens
- Generation of input files for RNA-seq and CNV data types
- Testing of RNA-seq and CNV modules specific functionality
- Testing of recipes

Changed

- More simplicity and abstraction for functions in main *ngs_toolkit.analysis.Analysis* class.

1.12.2 [0.19.3] - 2019-10-18

Changed

- Pin cffi version to fix bug in rpy2 with specific cffi version.

1.12.3 [0.19.2] - 2019-10-13

Added

- New module `ngs_toolkit.demo` which generates random data as PEP-formatted projects
- New `ngs_toolkit.recipes.generate_project` recipe to generate a new project using CLI
- New normalization method: variance stabilizing transformation (VST) available
- Add function to run `ngs_toolkit.recipes.ngs_analysis` recipe from initalized analysis object
- Add `distributed` mode to *ngs_toolkit.atacseq.ATACSeqAnalysis.measure_coverage()* using the new coverage recipe
- New `ngs_toolkit.recipes.coverage` recipe for calculating the coverage of a BAM file in BED regions
- Docs: usage of `sphinx-issues` for connecting to issue tracking and `sphinx-argparse` for automatic documentation of CLI of recipes

Changed

- Generator of random data now based on proper negative-binomial model
- Test suite now uses `ngs_toolkit.demo` module.
- change default of *ngs_toolkit.analysis.Analysis.differential_analysis()* to `filter_support=False`.
- fix boundary passing bug in subtract_principal_component
- Adopt Keep a Changelog changelog style.

1.12.4 [0.19.1] - 2019-10-09

Added

- Add `save` and `assign` arguments to *ngs_toolkit.atacseq.ATACSeqAnalysis.get_consensus_sites()*.
- New `ngs_toolkit.recipes.coverage`

Changed

- Stopped special handling reading of `matrix_norm` in *ngs_toolkit.analysis.Analysis.load_data()*. This now assumes a non-MultiIndex dataframe; fix #59.
- Change default of *ngs_toolkit.analysis.Analysis.annotate_samples()* `save` and `assign` arguments to `False`.
- *ngs_toolkit.analysis.Analysis.remove_factor_from_matrix()* now drops features with no variance.
- Change `filter_mito_chr` to `filter_chroms` argument of *ngs_toolkit.atacseq.ATACSeqAnalysis.get_consensus_sites()* in order to allow filtering arbitrary chromosomes out from consensus sites using a regex pattern. Supports multiple patterns by using the "|" operator.
- Much more efficient algorithm underlying *ngs_toolkit.atacseq.ATACSeqAnalysis.get_consensus_sites()* with speedup times up to 20x.
- Change method to compute coverage for *ngs_toolkit.atacseq.ATACSeqAnalysis.measure_coverage()* with `distributed=True` from `bedtools coverage` to `ngs_toolkit.recipes.coverage`. This has the following advantages: less reliance on bedtools; outputing a result for each region; same function as in serial mode.
- *ngs_toolkit.utils.count_reads_in_intervals()* now outputs coverage for every chromosome, outputs number of errors to log.
- Fix bug #61 - missing time parameter for enrichr job.
- Pin `yacman` version to 0.6.0.

1.12.5 [0.18.1] - 2019-10-03

Added

- Add `overwrite` argument to `ngs_toolkit.analysis.Analysis.measure_coverage()`.

Changed

- Fix #60: building of confusion matrix for Fisher Test in *ngs_toolkit.analysis.Analysis.differential_overlap()*.
- Use `-sorted` argument to `bedtools coverage` in `ngs_toolkit.analysis.Analysis.measure_coverage()` for fast and low-memory computing of coverage of BAM file in BED file when `distributed=True`.
- Set `setuptools_scm`, `requests`, `rpy2` and `natsort` versions.
- Extensive updates to documentation

1.12.6 [0.17.6] - 2019-09-25

Added

- Using `setuptools_scm` for semantic versioning.
- More testing of DESeq2 functionality.

Changed

- Removed `_version.py` file due to `setuptools_scm` adoption. API does not change though.
- Fixed continuous deployment in Travis.
- Dockerfile

1.12.7 [0.17.3] - 2019-09-24

Changed

- Fixed continuous deployment in Travis.

1.12.8 [0.17.2] - 2019-09-23

Changed

- Always display `ngs_toolkit` version in html report even if `pip_versions=False`.
- Pretty README on PyPI by specifying `long_description_content_type="text/markdown"` on setup.py.

1.12.9 [0.17.1] - 2019-09-23

Added

- Continuous deployment through Travis.
- Gitpod configuration
- Functionality to test whether `bedtools` version is acceptable.

- *ngs_toolkit.analysis.Analysis.get_sample_annotation()* for convinient getting of a current sample annotation based on *sample_attributes* and *group_attributes* given to `ngs_toolkit`.
- Add `deseq_kwargs` argument to *ngs_toolkit.analysis.Analysis.differential_analysis()* to allow passing of arguments to DESeq2 main function.
- Add functionality to repeat API call to `BioMart` in *ngs_toolkit.general.query_biomart()* with `max_api_retries` argument.

Changed

- Switched from custom install of non-Python requirements to Debian packages
- Required bedtools version is now 2.17.1
- Abstraction of `ngs_toolkit.decorators.check_organism_genome()` and `ngs_toolkit.decorators.check_has_sites()` to `ngs_toolkit.decorators.check_has_attributes()` which now accepts arguments.
- Add `save`, `assign` and `output_prefix` to *ngs_toolkit.analysis.Analysis.get_matrix_stats()*, *ngs_toolkit.analysis.Analysis.annotate_features()*, *ngs_toolkit.atacseq.ATACSeqAnalysis.get_peak_gene_annotation()* *ngs_toolkit.atacseq.ATACSeqAnalysis.get_peak_genomic_location()*, *ngs_toolkit.atacseq.ATACSeqAnalysis.get_peak_chromatin_state()*
- Set default arguments of *ngs_toolkit.analysis.Analysis.annotate_samples()* to `False`.
- Revamp of *ngs_toolkit.atacseq.ATACSeqAnalysis.plot_peak_characteristics()* with specific tests, but backward compatible API call.
- Avoid `matplotlib` version 3.1.1 due to bug manifesting on seaborn. Requirement now set to matplotlib>=2.1.1,<3.1.1.

1.12.10 [0.16.1] - 2019-09-04

Changed

- Fix bug in setup.py

1.12.11 [0.16] - 2019-09-04

Added

- Dockerfile

Changed

- Fixed bug in general.get_genomic_context due to a bug in the timestamping workflow
- Various fixes of timestamping and html reporting functionality
- Distributing tests with library for more portable testing
- Move rpy2 requirement to mandatory
- Make test data cases smaller for faster CI

1.12.12 [0.14] - 2019-08-28

Added

- Add recording of analysis outputs under Analysis.output_files
- Add timestamping of table and figure Analysis outputs
- Add HTML report with continuous generation
- *ngs_toolkit.analysis.Analysis.remove_factor_from_matrix()* for Combat removal of batch effects
- More documentation on installation particularly for setting up in a Conda environment

Changed

- Now testing on Ubuntu 18.04 for Python 3.6 and 3.7 only.
- CNV module updated
- recipe updated

1.12.13 [0.12] - 2019-08-12

Changed

- change of unsupervised_analysis API call: homogeneization with remaining functions
- optional saving of embeddings and loadings of PCA and manifolds in unsupervised_analysis

1.12.14 [0.11] - 2019-08-08

Added

- support for additional keyword arguments passed to Project initialization when using *from_pep*

Changed

- adapt to latest versions of pepkit stack
- better colouring of sample group levels in get_level_colors

1.12.15 [0.10]

Added

- `normalize_by_background` function to ChIPSeqAnalysis to normalize over background samples

Changed

- revamp RNASeqAnalysis
- adapt ChIPSeqAnalysis to new API
- fix logger due to accidental deactivation

1.12.16 [0.9]

Added

- add annotate_matrix to call both above.

Changed

- rename annotate to annotate_features and annotate_with_sample_metadata to annotate_samples

1.12.17 [0.8]

Changed

- usage of the same interpreter running ngs_toolkit to run jobs
- revamp recipes, usage of recipes for common work functions that run in distributed mode
- allow import of classes from root of library.

1.12.18 [0.7]

Added

- implement running of local or distributed jobs using `divvy`

1.12.19 [0.6]

Changed

- rename merge_table to sample_subannotation

1.12.20 [0.5]

Changed

- major API changes
- implementing only two types of matrices: matrix_raw, matrix_norm
- unify normalization methods, each overwrites matrix_norm and sets flag with name of method used

1.12.21 [0.2.1] - 2018-12-13

- minor:
  - change default directory for enrichment results
  - add class method to overwrite Sample object representation
  - add configuration to merge_signal recipe
  - add graphics functions
  - add optional requirements for single cell analysis
  - add possibility of different prefixes when collecting enrichments
  - remove requirement of some comparison_table and attributes_to_plot arguments
  - remove obsolete functions
  - more powerful Analysis objects by leveraging on known Project attributes
  - simplify plot of number of differential regions per comparison in plot_differential
- bug fixes:
  - fix pipy install on Python 3: requirements.txt is now distributed with package
  - update merge_signal recipe - fix bug when grouping samples by only one attribute
  - better error catching
  - fix LOLA output collection saving when running in serial mode
  - fix choice of common p-value color overlay to plot in plot_differential
  - fix creating job in merge_signal recipe
  - fix invalid yaml in configs
  - fix mistake in requirements for peppy
  - fix some security issues

1.12.22 [0.1.6.0] - 2018-12-05

- major:
  - New CNV module
  - many fixes and improvements to run various enrichment analysis in serial
  - add specific attributes to classes - this will be the basis of the new API revamp
  - add support for running specific steps of enrichment analysis
  - better utf8 encoding support to all Enrichr inputs/outputs
  - add support for plotting 1 attribute in unsupervised_analysis
  - add support for limma regression without covariates; more help messages
  - fix bug in plot_differential when plotting scatter with colours per p-value
  - improved general.query_biomart to handle fields with multiple values
  - update requirements

- minor:
  - now plotting MA, scatter and volcano plots even if there are no significant variables
  - plot log variance in PCA
  - better docstring styling (in progress)
  - plotting signed p-value heatmap
  - support case when only one feature is differential
  - add option to turn on independnent filtering in DESeq2
  - add y log scale to p-value and fold-change distplots
  - homogeneize range of p-value colouring of scatter, volcano and MA plots across comparisons - new colormap
  - better handling of missing comparisons in general.plot_differential
  - better plotting of plot_differential p-values
  - fix example config to correct paths
  - add verbosity to manager programs
  - reporting more info for plot_differential

1.12.23 [0.1.5.1] - 2018-11-25

- add config file support for better system-independent operation (specially for enrichment analysis)
- add logger through "logging" library
- add batch effect correction with limma
- add GREAT parser
- add colouring by p-value for plot_differential
- add set n. of PCs to calculate to PCA
- add better colorbars
- add serial processing of peak commands as option for ChIP-seq peak calling

1.12.24 [0.1.4.2] - 2018-10-29

- fix important lack of ngs_toolkit.recipes module in setup.py: recipes should now be correctly added to $PATH
- fix and add full support to comparison_table in recipes.ngs_analysis
- add region_set_frip recipe
- add merge_signal recipe
- add PEP badge
- ngs_toolkit.general:
  - fix when general.collect_differential_enrichment reads an empty motif enrichment file
  - delete existing files if existing in general.homer_combine_motifs
  - report unmatched differnetial and total features in general.plot_differential

  - general.collect_differential_analysis now sets index of differential_results dataframe correctly
  - add more manifold methods to general.unsupervised_analysis: Isomap, LocallyLinearEmbedding, SpectralEmbedding, TSNE in addition to MDS (and PCA)
  - add ChIP-seq as a valid data type to general.unsupervised_analysis
  - add standardization to parameters of general.unsupervised_analysis
  - add level labels to group labeling of general.unsupervised_analysis and general.plot_differential
  - new default color palletes
  - add option of matching motifs only to known vertebrate in general.homer_consensus
  - add heatmap plotting of enrichment over background for homer consensus in plot_differential_enrichment
  - change default output_dir of general.unsupervised_analysis
  - add more descriptive labels to tqdm loops;
  - add CPU/mem parametrization of general.differential_analysis when running in job mode
  - quick fix for pypiper.ngstk >= 0.6 compatibility (tabs vs spaces) in general.differential_analysis - needs revision
  - resolve pandas warnings of setting without .loc
- ngs_toolkit.chipseq:
  - add function to filter_peaks
  - add more descriptive labels to tqdm loops;
  - fix overaping peaks calling job files in chipseq.summarize_peaks_from_comparisons
- ngs_toolkit.atacseq:
  - add more descriptive labels to tqdm loops;

1.12.25 [0.1.4] - 2018-09-25

- Update to peppy version 0.9.1
- fixes/improvements:
  - add fold enrichment vs p-value plots in homer_consensus plot_differential_enrichments()
  - add index name to DESeq2 CSV output; fix import on homer_combine_motifs
  - add recipes to command-line entry in setup.py; remove cPickle import; better style
  - add scatterplots to enrichr type of data in plot_differential_enrichment
  - add self.data_type to Analysis objects
  - added "homer_consensus" as one type of possible enrichment in plot_differential_enrichment, related to issue 21
  - crunch landscape bad score for __init__;
  - default color range of numeric values in get_level_colors to min-max
  - fix issue 19
  - fix issue 24; general.project_to_geo file referencing
  - implement homer consensus motifs as in issue 21; add collectiong and plotting of homer enrichmnts

  - moved annotate_with_sample_metadata to base Analysis class
  - tested qnorm implementations; switched to Python implementation, close issue 12
- documentation:
  - docs for the region_set_frip, merge_signal and call_peaks recipes

1.12.26 [0.1.3.6] - 2018-08-05

- add two new recipes:
  - region_set_frip: calculate FRiP in a consensus region set of interest for all samples (rsFRiP)
  - merge_signal: create merged signal data for samples sharing specific attributes. Creates BAM files, bigWig files, and BAM files for nucleosomal and nucleosomal-free reads based on fragment size
- trackmanager:
  - Fix issue #16: trackmanager output indentation
  - add default attributes to specified in project_config.group_attributes or otherwise to ['sample_name']
  - fix empty subGroups in UCSC trackDb file
  - remove required attributes if no value is found
- Fix issue #20: len(attributes_to_plot) in general.unsupervised_analysis can be 1 now
- add Makefile to upload to Pypi
- update looper template folder of projectmanager
- add default time to longq in analysis_job task in projectmanager Makefile
- add unbuferred output to ngs_analysis recipe
- add atacseq.get_gene_level_changes
- improve atacseq.get_gene_level_accessibility
- add 2D support to general.signed_mean

1.12.27 [0.1.3.5.3b] - 2018-06-12

- Fixes:
  - general.deseq_analysis: fix hyphen character conversion; better contrasts for DESeq2

1.12.28 [0.1.3.5.3] - 2018-05-31

- Fixes:
  - projectmanager: fix Makefile creation
  - ngs_analysis recipe: change selection of samples on "pass_qc"; do differential_overlap only when >1 comparison

1.12.29 [0.1.3.5.2] - 2018-05-30

- Fixes:
  - trackmanager: trackHeight attribute typo making tracks have 128 pixels height
  - trackmanager: sampleGroup attribute indentation
- New:
  - general.plot_differential: center divergent heatmaps on 0 in, add sorted heatmaps
  - General *ngs_analysis* recipe for general analysis of NGS projects.

1.12.30 [0.1.3.5] - 2018-05-15

- New:
  - Extended documentation
  - Command-line interface (CLI) based on sub-commands for `projectmanager`.
  - Recipes: scripts which `projectmanager` can run.
  - General *ngs_analysis* recipe for general analysis of NGS projects.

1.13 Indices and tables

- genindex
- modindex
- search

CHAPTER

TWO

LINKS

- Documentation: http://toolkit.readthedocs.io/
- Issues and source code: https://github.com/afrendeiro/toolkit

PYTHON MODULE INDEX

n

INDEX