deepTools Documentation

Release 3.4.3

Fidel Ramírez, Friederike Dündar, Björn Grüning, Thomas Manke

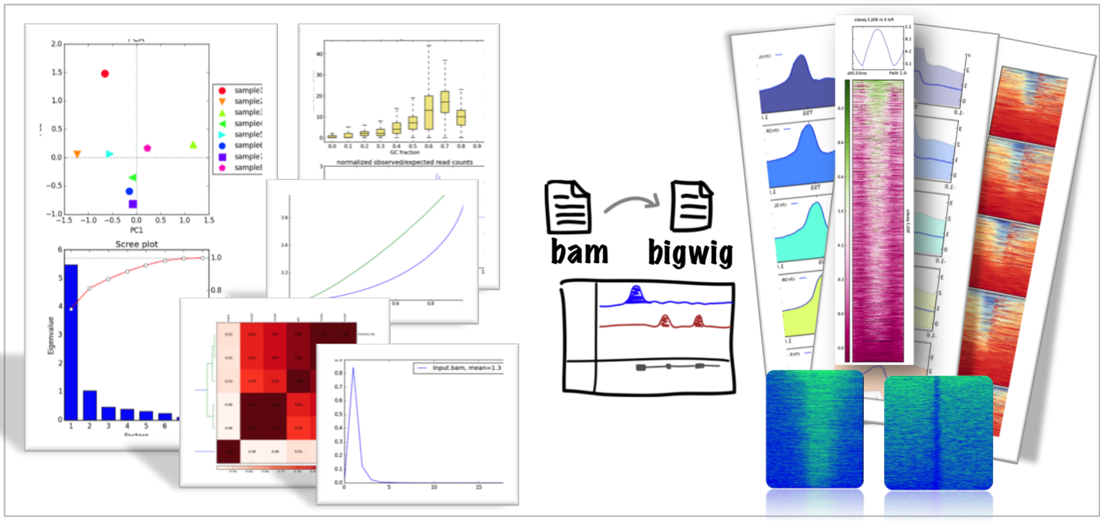
Aug 12, 2020

Contents

1 Contents: 3

Python Module Index 107

Index 109



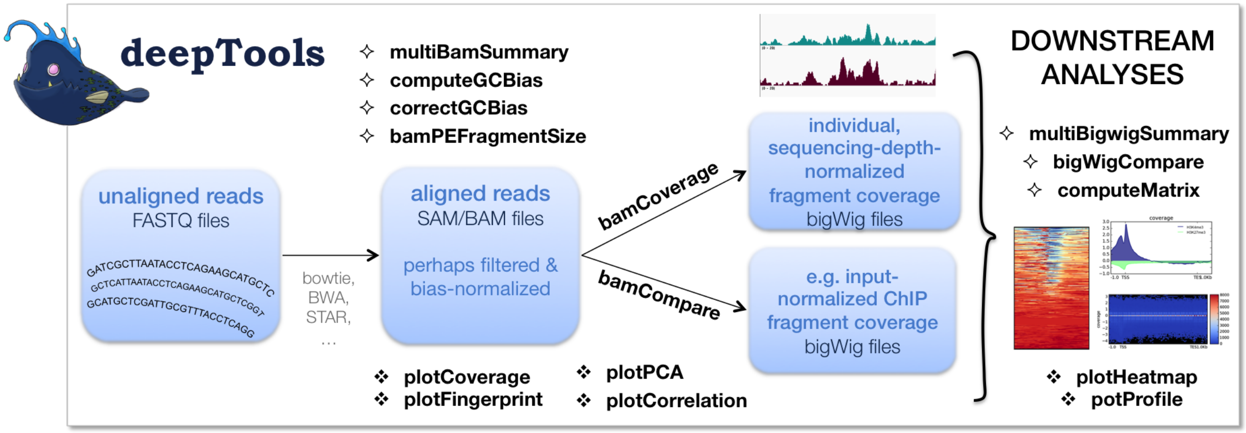
QUALITY CHECKS – FORMAT CONVERSION & NORMALIZATION – PLOTTING

deepTools is a **suite of python tools** particularly developed for the efficient analysis of high-throughput sequencing data, such as ChIP-seq, RNA-seq or MNase-seq.

There are 3 ways for using deepTools:

- **Galaxy usage** – our public deepTools Galaxy server let's you use the deepTools within the familiar Galaxy framework without the need to master the command line
- **command line usage** – simply download and install the tools (see *Installation* and *The tools*)
- **API** – make use of your favorite deepTools modules in your own python programs (see *deepTools API*)

The flow chart below depicts the different tool modules that are currently available.


deepTools
✧ multiBamSummary
✧ computeGCBias
✧ correctGCBias
✧ bamPEFragmentSize
unaligned reads
FASTQ files
GATCGCTTAATACCTCAGAAGCATGCTC
GCTCATTAATACCTCAGAAGCATGCTCGGT
GCATGCTCGATTGCGTTTACCTCAGG
bowtie,
BWA,
STAR,
...
aligned reads
SAM/BAM files
perhaps filtered &
bias-normalized
bamCoverage
bamCompare
individual,
sequencing-depth-
normalized
fragment coverage
bigWig files
e.g. input-
normalized ChIP
fragment coverage
bigWig files
❖ plotCoverage
❖ plotFingerprint
❖ plotPCA
❖ plotCorrelation
DOWNSTREAM
ANALYSES
✧ multiBigwigSummary
✧ bigWigCompare
✧ computeMatrix
❖ plotHeatmap
❖ potProfile

If the file names in the figure mean nothing to you, please make sure to check our *Glossary of NGS terms*.

CHAPTER 1

Contents:

1.1 Installation

Remember – deepTools are available for **command line usage** as well as for **integration into Galaxy servers**!

- *Requirements*
- *Command line installation using* `pip`
- *Command line installation without* `pip`
- *Galaxy installation*
    - *Installation via Galaxy API (recommended)*
    - *Installation via web browser*
    - *Installation with Docker*

1.1.1 Requirements

- Python 2.7 or Python 3.x
- numpy >= 1.8.0
- scipy >= 0.17.0
- py2bit >= 0.1.0
- pyBigWig >= 0.2.1
- pysam >= 0.8
- matplotlib >= 1.4.0

The fastest way to obtain **Python 2.7 or Python 3.x together with numpy and scipy** is via the Anaconda Scientific Python Distribution. Just download the version that's suitable for your operating system and follow the directions for its installation. All of the requirements for deepTools can be installed in Anaconda with:

```
$ conda install -c bioconda deeptools
```

1.1.2 Command line installation using `pip`

Install deepTools using the following command:

```
$ pip install deeptools
```

All python requirements should be automatically installed.

If you need to specify a specific path for the installation of the tools, make use of *pip install*'s numerous options:

```
$ pip install --install-option="--prefix=/MyPath/Tools/deepTools2.0" git+https://
↪github.com/deeptools/deepTools.git
```

1.1.3 Command line installation without `pip`

You are highly recommended to use *pip* rather than these more complicated steps.

1. Install the requirements listed above in the "requirements" section. This is done automatically by *pip*.

2. Download source code

```
$ git clone https://github.com/deeptools/deepTools.git
```

or if you want a particular release, choose one from https://github.com/deeptools/deepTools/releases:

```
$ wget https://github.com/deeptools/deepTools/archive/1.5.12.tar.gz
$ tar -xzvf
```

3. install the source code (if you don't have root permission, you can set a specific folder using the `--prefix` option)

```
$ python setup.py install --prefix /User/Tools/deepTools2.0
```

1.1.4 Galaxy installation

deepTools can be easily integrated into a local Galaxy. All wrappers and dependencies are available in the Galaxy Tool Shed.

Installation via Galaxy API (recommended)

First generate an API Key for your admin user and run the the installation script:

```
$ python ./scripts/api/install_tool_shed_repositories.py \
        --api YOUR_API_KEY -l http://localhost/ \
        --url http://toolshed.g2.bx.psu.edu/ \
        -o bgruening -r <revision> --name suite_deeptools \
        --tool-deps --repository-deps --panel-section-name deepTools
```

The `-r` argument specifies the version of deepTools. You can get the latest revision number from the test tool shed or with the following command:

```
$ hg identify http://toolshed.g2.bx.psu.edu/repos/bgruening/suite_deeptools
```

You can watch the installation status under: Top Panel –> Admin –> Manage installed tool shed repositories

Installation via web browser

- go to the admin page
- select *Search and browse tool sheds*
- Galaxy tool shed –> Sequence Analysis –> deeptools
- install deeptools

Installation with Docker

The deepTools Galaxy instance is also available as a docker container, for those wishing to use the Galaxy framework but who also prefer a virtualized solution. This container is quite simple to install:

```
$ sudo docker pull quay.io/bgruening/galaxy-deeptools
```

To start and otherwise modify this container, please see the instructions on the docker-galaxy-stable github repository. Note that you must use *bgruening/galaxy-deeptools* in place of *bgruening/galaxy-stable* in the examples, as the deepTools Galaxy container is built on top of the galaxy-stable container.

Tip: For support or questions please make a post on Biostars. For feature requests or bug reports please open an issue on github.

deepTools Galaxy.	code @ github.

1.2 The tools

Note: With the release of deepTools 2.0, we renamed a couple of tools:

- **heatmapper** to tools/plotHeatmap
- **profiler** to tools/plotProfile
- **bamCorrelate** to tools/multiBamSummary
- **bigwigCorrelate** to tools/multiBigwigSummary
- **bamFingerprint** to tools/plotFingerprint.

For more, see *Changes in deepTools2.0*.

- *General principles*

  - *Parameters to decrease the run time*
  - *Filtering BAMs while processing*
- *Tools for BAM and bigWig file processing*
  - *tools/multiBamSummary*
  - *tools/multiBigwigSummary*
  - *tools/correctGCBias*
  - *tools/bamCoverage*
  - *tools/bamCompare*
  - *tools/bigwigCompare*
  - *tools/computeMatrix*
  - *tools/alignmentSieve*
- *Tools for QC*
  - *tools/plotCorrelation*
  - *tools/plotPCA*
  - *tools/plotFingerprint*
  - *tools/bamPEFragmentSize*
  - *tools/computeGCBias*
  - *tools/plotCoverage*
- *Heatmaps and summary plots*
  - *tools/plotHeatmap*
  - *tools/plotProfile*
  - *tools/plotEnrichment*
- *Miscellaneous*
  - *tools/computeMatrixOperations*
  - *tools/estimateReadFiltering*

tool	type	input files	main output file(s)	application
tools/multiBamSummary	data integration	2 or more BAM	interval-based table of values	perform cross-sample analyses of read counts –> plotCorrelation, plotPCA
tools/multiBigwigSummary	data integration	2 or more bigWig	interval-based table of values	perform cross-sample analyses of genome-wide scores –> plotCorrelation, plotPCA
tools/plotCorrelation	visualization	bam/multiBigwigSummary output	clustered heatmap	visualize the Pearson/Spearman correlation
tools/plotPCA	visualization	bam/multiBigwigSummary output	2 PCA plots	visualize the principal component analysis
tools/plotFingerprint	QC	2 BAM	1 diagnostic plot	assess enrichment strength of a ChIP sample
tools/computeGCBias	QC	1 BAM	2 diagnostic plots	calculate the exp. and obs. GC distribution of reads
tools/correctGCBias	QC	1 BAM, output from computeGCbias	1 GC-corrected BAM	obtain a BAM file with reads distributed according to the genome's GC content
tools/bamCoverage	normalization	BAM	bedGraph or bigWig	obtain the normalized read coverage of a single BAM file
tools/bamCompare	normalization	2 BAM	bedGraph or bigWig	normalize 2 files to each other (e.g. log2ratio, difference)
tools/computeMatrix	data integration	1 or more bigWig, 1 or more BED	zipped file for plotHeatmap or plotProfile	compute the values needed for heatmaps and summary plots
tools/estimateReadFiltering	information	1 or more BAM files	table of values	estimate the number of reads filtered from a BAM file or files
tools/alignmentSieve	QC	1 BAM file	1 filtered BAM or BEDPE file	filters a BAM file based on one or more criteria
tools/plotHeatmap	visualization	computeMatrix output	heatmap of read coverages	visualize the read coverages for genomic regions
tools/plotProfile	visualization	computeMatrix output	summary plot ("meta-profile")	visualize the average read coverages over a group of genomic regions
tools/plotCoverage	visualization	1 or more BAM	2 diagnostic plots	visualize the average read coverages over sampled genomic positions
tools/bamPEFragmentSize	information	1 BAM	text with paired-end fragment length	obtain the average fragment length from paired ends
tools/plotEnrichment	visualization	1 or more BAM and 1 or more BED/GTF	A diagnostic plot	plots the fraction of alignments overlapping the given features
tools/computeMatrixOperations	miscellaneous	1 or more BAM and 1 or more BED/GTF	A diagnostic plot	plots the fraction of alignments overlapping the given features

1.2.1 General principles

A typical deepTools command could look like this:

```
$ bamCoverage --bam myAlignedReads.bam \
--outFileName myCoverageFile.bigWig \
--outFileFormat bigwig \
--fragmentLength 200 \
--ignoreDuplicates \
--scaleFactor 0.5
```

You can always see all available command-line options via –help:

```
$ bamCoverage --help
```

- Output format of plots should be indicated by the file ending, e.g. `MyPlot.pdf` will return a pdf file, `MyPlot.png` a png-file
- All tools that produce plots can also output the underlying data - this can be useful in cases where you don't like the deepTools visualization, as you can then use the data matrices produced by deepTools with your favorite plotting tool, such as R
- The vast majority of command line options are also available in Galaxy (in a few cases with minor changes to their naming).

Parameters to decrease the run time

- **`numberOfProcessors` - Number of processors to be used**

  For example, setting `--numberOfProcessors 10` will split up the workload internally into 10 chunks, which will be processed in parallel.
- **`region` - Process only a single genomic region.** This is particularly useful when you're still trying to figure out the best parameter setting. You can focus on a certain genomic region by setting, e.g., `--region chr2` or `--region chr2:100000-200000`

These parameters are optional and available throughout almost all deepTools.

Filtering BAMs while processing

Several deepTools modules allow for efficient processing of BAM files, e.g. `bamCoverage` and `bamCompare`. We offer several ways to filter those BAM files on the fly so that you don't need to pre-process them using other tools such as samtools

- **`ignoreDuplicates`** Reads with the same orientation and start position will be considered only once. If reads are paired, the mate is also evaluated
- **`minMappingQuality`** Only reads with a mapping quality score of at least this are considered
- **`samFlagInclude`** Include reads based on the SAM flag, e.g. `--samFlagInclude 64` gets reads that are first in a pair. For translating SAM flags into English, go to: https://broadinstitute.github.io/picard/explain-flags.html
- **`samFlagExclude`** Exclude reads based on the SAM flags - see previous explanation.

These parameters are optional and available throughout deepTools.

Note: In version 2.3 we introduced a sampling method to correct the effect of filtering when normalizing using `bamCoverage` or `bamCompare`. For previous versions, if you know that your files will be strongly affected by the filtering of duplicates or reads of low quality then consider removing those reads *before* using `bamCoverage` or `bamCompare`, as the filtering by deepTools is done *after* the scaling factors are calculated!

1.2.2 Tools for BAM and bigWig file processing

tools/multiBamSummary

tools/multiBigwigSummary

tools/correctGCBias

tools/bamCoverage

tools/bamCompare

tools/bigwigCompare

tools/computeMatrix

tools/alignmentSieve

1.2.3 Tools for QC

tools/plotCorrelation

tools/plotPCA

tools/plotFingerprint

tools/bamPEFragmentSize

tools/computeGCBias

tools/plotCoverage

1.2.4 Heatmaps and summary plots

tools/plotHeatmap

tools/plotProfile

tools/plotEnrichment

1.2.5 Miscellaneous

tools/computeMatrixOperations

tools/estimateReadFiltering

deepTools Galaxy.	code @ github.

1.3 Advanced features

Some of the features of deepTools are not self-explanatory. Below, we provide links to longer expositions on these more advanced features:

- feature/blacklist
- feature/metagene
- feature/read_extension
- feature/unscaled_regions
- feature/read_offsets
- feature/deepBlue
- feature/plotFingerprint_QC_metrics
- feature/plotly
- feature/effectiveGenomeSize

deepTools Galaxy.	code @ github.

1.4 Example usage

- *How we use deepTools for ChIP-seq analyses*

1.4.1 Step-by-step protocols

This section should give you an overview of how to do many common tasks. We're using **screenshots from Galaxy** here. If you're using the command-line version you can easily follow the given examples since the vast majority of parameters is either indicated in Galaxy, too. Otherwise, just type the program name and the help option (e.g. `/deepTools/bin/bamCoverage --help`), which will show you all the parameters and options available. Alternatively, you can follow the respective link to the tool documentation here on readthedocs.

Note: For support or questions please post to Biostars. For bug reports and feature requests please open an issue <on github.

All protocols assume that you have uploaded your files into a Galaxy instance with a deepTools installation, e.g., deepTools Galaxy. If you need help to get started with Galaxy in general, e.g. to upload your data, see *Using deepTools within Galaxy* and *Data import into Galaxy*.

Tip: If you would like to try out the protocols with **sample data**, go to deepTools Galaxy –> "Shared Data" –> "Data Libraries" –> "deepTools Test Files". Simply select BED/BAM/bigWig files and click, "to History". You can also download the test data sets to your computer by clicking "Download" at the top.

How to do. . . ?

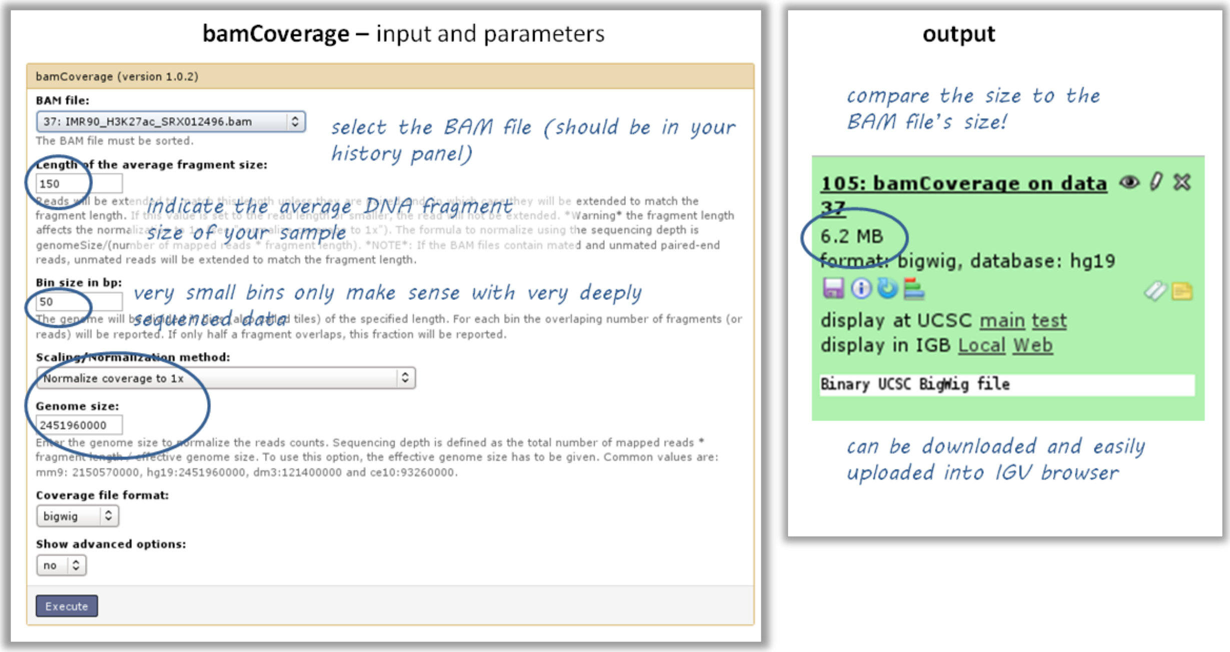
- *QC and data processing*
  - *I have downloaded/received a BAM file - how do I generate a file I can look at in a genome browser?*
  - *How can I assess the reproducibility of my sequencing replicates?*
  - *How do I know whether my sample is GC biased? And if it is, how do I correct for it?*
  - *How do I get an input-normalized ChIP-seq coverage file?*
  - *How can I compare the ChIP strength for different ChIP experiments?*
- *Heatmaps and summary plots*
  - *How do I get a (clustered) heatmap of sequencing-depth-normalized read coverages around the transcription start site of all genes?*
  - *How can I compare the average signal for X-specific and autosomal genes for 2 or more different sequencing experiments?*
    - *How to obtain a BED file for X chromosomal and autosomal genes each*
    - *Compute the average values for X and autosomal genes*

QC and data processing

I have downloaded/received a BAM file - how do I generate a file I can look at in a genome browser?

- tool: tools/bamCoverage
- input: your *BAM* file with aligned reads

Of course, you could also look at your BAM file in the genome browser. However, generating a bigWig file of read coverages will drastically reduce the size of the file, it also allows you to normalize the coverage to 1x sequencing depth, which makes a visual comparison of multiple files more feasible.



bamCoverage – input and parameters

bamCoverage (version 1.0.2)

BAM file: 37: IMR90_H3K27ac_SRX012496.bam

The BAM file must be sorted.

select the BAM file (should be in your history panel)

Length of the average fragment size: 150

Reads will be extended to match this length unless they are paired-end, in which case they will be extended to match the fragment length. If this value is set to the read length or smaller, the read will not be extended. *Warning* the fragment length affects the normalization to 1x (see "normalize coverage to 1x"). The formula to normalize using the sequencing depth is genomeSize/(number of mapped reads * fragment length). *NOTE*: If the BAM files contain mated and unmated paired-end reads, unmated reads will be extended to match the fragment length.

indicate the average DNA fragment size of your sample

Bin size in bp: 50

The genome will be divided in bins (also called tiles) of the specified length. For each bin the overlaping number of fragments (or reads) will be reported. If only half a fragment overlaps, this fraction will be reported.

very small bins only make sense with very deeply sequenced data

Scaling/Normalization method: Normalize coverage to 1x

Genome size: 2451960000

Enter the genome size to normalize the reads counts. Sequencing depth is defined as the total number of mapped reads * fragment length / effective genome size. To use this option, the effective genome size has to be given. Common values are: mm9: 2150570000, hg19:2451960000, dm3:121400000 and ce10:93260000.

Coverage file format: bigwig

Show advanced options: no

output

compare the size to the BAM file's size!

105: bamCoverage on data 37

6.2 MB

format: bigwig, database: hg19

display at UCSC main test

display in IGB Local Web

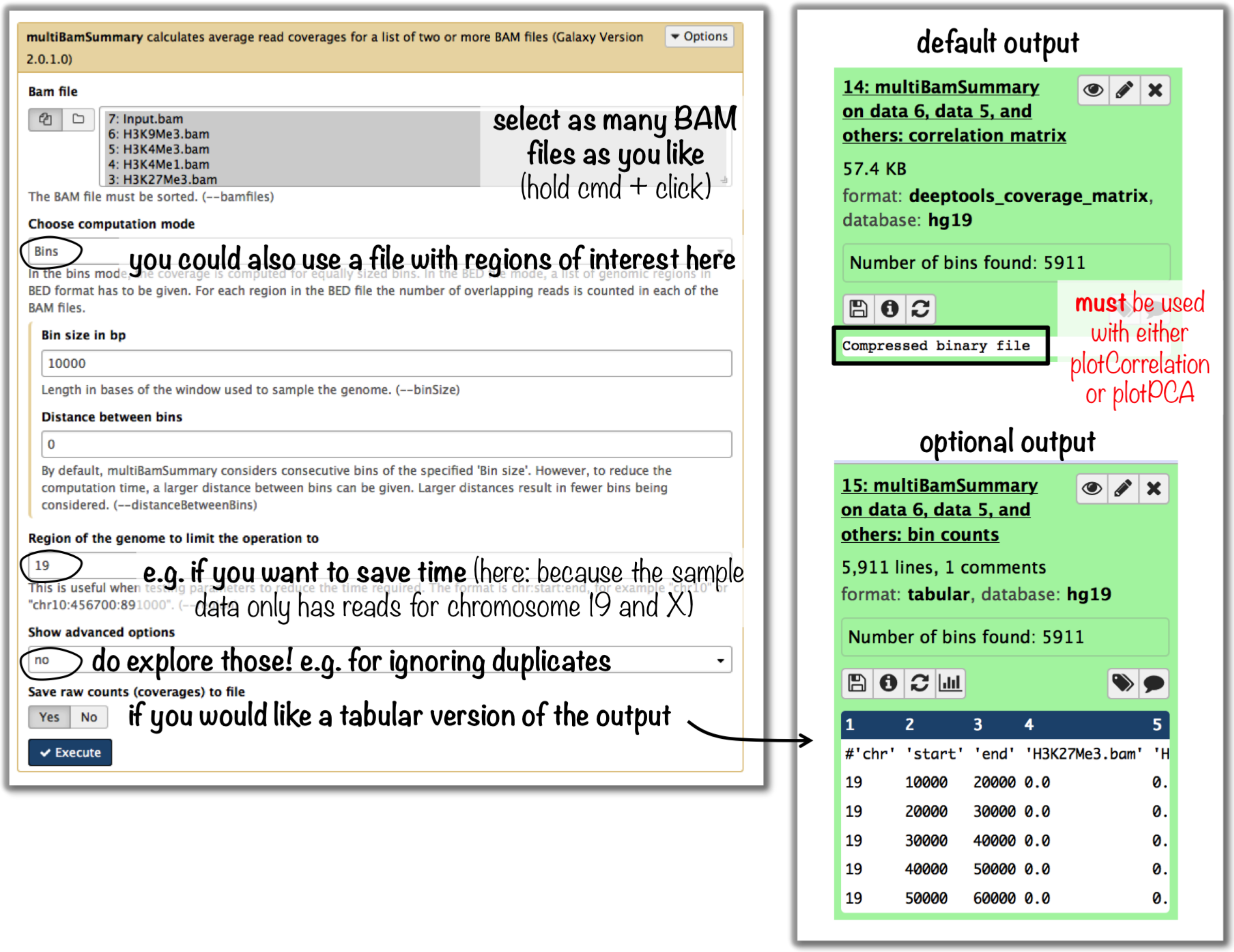
Binary UCSC BigWig file

can be downloaded and easily uploaded into IGV browser

How can I assess the reproducibility of my sequencing replicates?

Typically, you're going to be interested in the correlation of the read coverages for different replicates and different samples. What you want to see is that replicates should correlate better than non-replicates. The ENCODE consortium recommends that *for messenger RNA, (. . .) biological replicates [should] display 0.9 correlation for transcripts/features*. For more information about correlation calculations, see the background description for tools/plotCorrelation.

- tools: tools/multiBamSummary followed by tools/plotCorrelation
- **input: BAM files**
    - you can compare as many samples as you want, though the more you use the longer the computation will take



multiBamSummary calculates average read coverages for a list of two or more BAM files (Galaxy Version 2.0.1.0)

Bam file

7: Input.bam
6: H3K9Me3.bam
5: H3K4Me3.bam
4: H3K4Me1.bam
3: H3K27Me3.bam

select as many BAM files as you like (hold cmd + click)

The BAM file must be sorted. (--bamfiles)

Choose computation mode

Bins

you could also use a file with regions of interest here

In the bins mode, the coverage is computed for equally sized bins. In the BED file mode, a list of genomic regions in BED format has to be given. For each region in the BED file the number of overlapping reads is counted in each of the BAM files.

Bin size in bp

10000

Length in bases of the window used to sample the genome. (--binSize)

Distance between bins

0

By default, multiBamSummary considers consecutive bins of the specified 'Bin size'. However, to reduce the computation time, a larger distance between bins can be given. Larger distances result in fewer bins being considered. (--distanceBetweenBins)

Region of the genome to limit the operation to

19

e.g. if you want to save time (here: because the sample data only has reads for chromosome 19 and X)

This is useful when testing parameters to reduce the time required. The format is chr:start:end, for example "chr10" or "chr10:456700:891000". (--

Show advanced options

no

do explore those! e.g. for ignoring duplicates

Save raw counts (coverages) to file

[✓ Yes] No

if you would like a tabular version of the output

default output

14: multiBamSummary on data 6, data 5, and others: correlation matrix

57.4 KB

format: **deeptools_coverage_matrix**, database: **hg19**

Number of bins found: 5911

Compressed binary file

must be used with either plotCorrelation or plotPCA

optional output

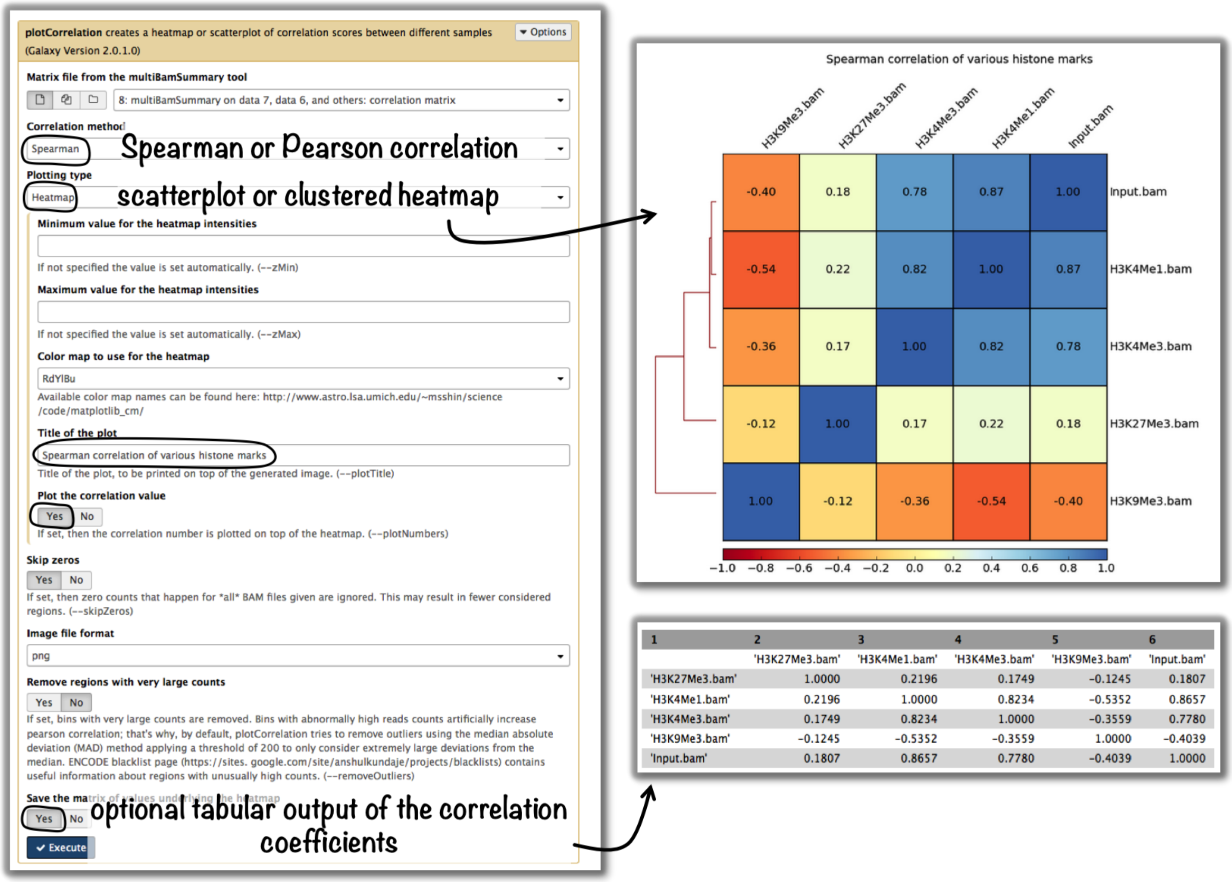
15: multiBamSummary on data 6, data 5, and others: bin counts

5,911 lines, 1 comments

format: **tabular**, database: **hg19**

Number of bins found: 5911

```
1      2       3      4              5
#'chr'  'start'  'end'  'H3K27Me3.bam'  'H
19      10000    20000  0.0             0.
19      20000    30000  0.0             0.
19      30000    40000  0.0             0.
19      40000    50000  0.0             0.
19      50000    60000  0.0             0.
```



Options
Execute

plotCorrelation creates a heatmap or scatterplot of correlation scores between different samples (Galaxy Version 2.0.1.0)

Matrix file from the multiBamSummary tool
8: multiBamSummary on data 7, data 6, and others: correlation matrix

Correlation method
Spearman

Spearman or Pearson correlation

Plotting type
Heatmap

scatterplot or clustered heatmap

Minimum value for the heatmap intensities
If not specified the value is set automatically. (--zMin)

Maximum value for the heatmap intensities
If not specified the value is set automatically. (--zMax)

Color map to use for the heatmap
RdYlBu
Available color map names can be found here: http://www.astro.lsa.umich.edu/~msshin/science/code/matplotlib_cm/

Title of the plot
Spearman correlation of various histone marks
Title of the plot, to be printed on top of the generated image. (--plotTitle)

Plot the correlation value
[✓ Yes] No
If set, then the correlation number is plotted on top of the heatmap. (--plotNumbers)

Skip zeros
[✓ Yes] No
If set, then zero counts that happen for *all* BAM files given are ignored. This may result in fewer considered regions. (--skipZeros)

Image file format
png

Remove regions with very large counts
Yes [✓ No]
If set, bins with very large counts are removed. Bins with abnormally high reads counts artificially increase pearson correlation; that's why, by default, plotCorrelation tries to remove outliers using the median absolute deviation (MAD) method applying a threshold of 200 to only consider extremely large deviations from the median. ENCODE blacklist page (https://sites. google.com/site/anshulkundaje/projects/blacklists) contains useful information about regions with unusually high counts. (--removeOutliers)

Save the matrix of values underlying the heatmap
[✓ Yes] No

optional tabular output of the correlation coefficients

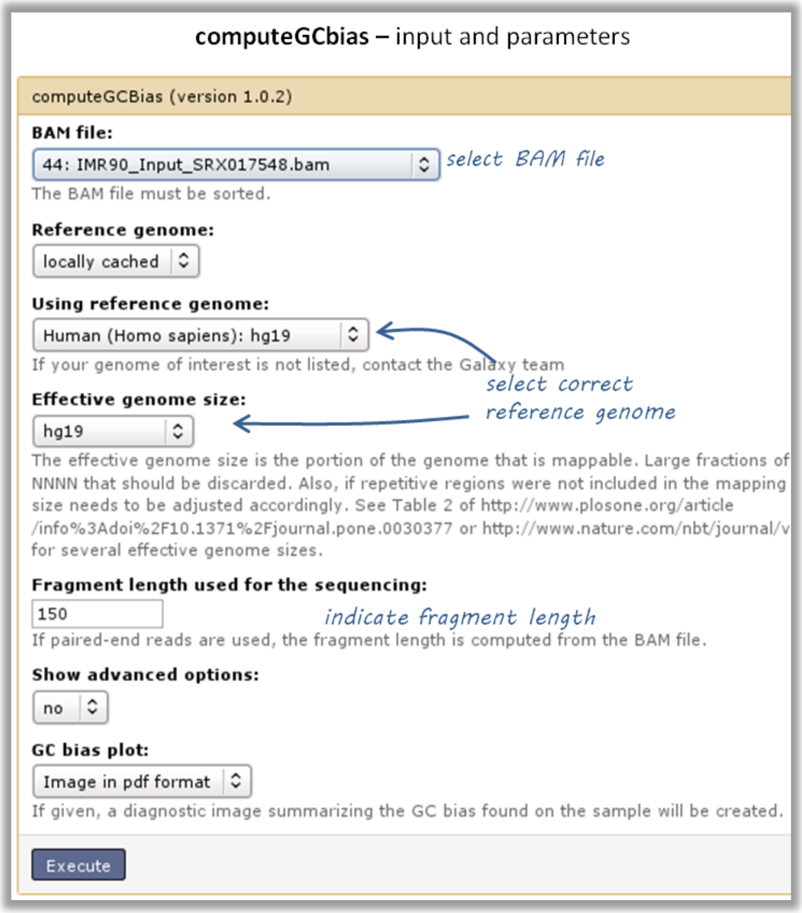
Spearman correlation of various histone marks
H3K9Me3.bam
H3K27Me3.bam
H3K4Me3.bam
H3K4Me1.bam
Input.bam
-0.40 0.18 0.78 0.87 1.00 Input.bam
-0.54 0.22 0.82 1.00 0.87 H3K4Me1.bam
-0.36 0.17 1.00 0.82 0.78 H3K4Me3.bam
-0.12 1.00 0.17 0.22 0.18 H3K27Me3.bam
1.00 -0.12 -0.36 -0.54 -0.40 H3K9Me3.bam
−1.0 −0.8 −0.6 −0.4 −0.2 0.0 0.2 0.4 0.6 0.8 1.0

1	2	3	4	5	6
	'H3K27Me3.bam'	'H3K4Me1.bam'	'H3K4Me3.bam'	'H3K9Me3.bam'	'Input.bam'
'H3K27Me3.bam'	1.0000	0.2196	0.1749	-0.1245	0.1807
'H3K4Me1.bam'	0.2196	1.0000	0.8234	-0.5352	0.8657
'H3K4Me3.bam'	0.1749	0.8234	1.0000	-0.3559	0.7780
'H3K9Me3.bam'	-0.1245	-0.5352	-0.3559	1.0000	-0.4039
'Input.bam'	0.1807	0.8657	0.7780	-0.4039	1.0000

Tip: If you would like to do a similar analysis based on bigWig files, use the tool `multiBigwigSummary` instead.

How do I know whether my sample is GC biased? And if it is, how do I correct for it?

- input: *BAM* file
- use the tool tools/computeGCBias on that BAM file (default settings, just **make sure your reference genome and genome size are matching**)



computeGCbias – input and parameters

computeGCBias (version 1.0.2)

BAM file:

44: IMR90_Input_SRX017548.bam

The BAM file must be sorted.

Reference genome:

locally cached

Using reference genome:

Human (Homo sapiens): hg19

If your genome of interest is not listed, contact the Galaxy team

Effective genome size:

hg19

The effective genome size is the portion of the genome that is mappable. Large fractions of NNNN that should be discarded. Also, if repetitive regions were not included in the mapping size needs to be adjusted accordingly. See Table 2 of http://www.plosone.org/article /info%3Adoi%2F10.1371%2Fjournal.pone.0030377 or http://www.nature.com/nbt/journal/v for several effective genome sizes.

Fragment length used for the sequencing:

150

If paired-end reads are used, the fragment length is computed from the BAM file.

Show advanced options:

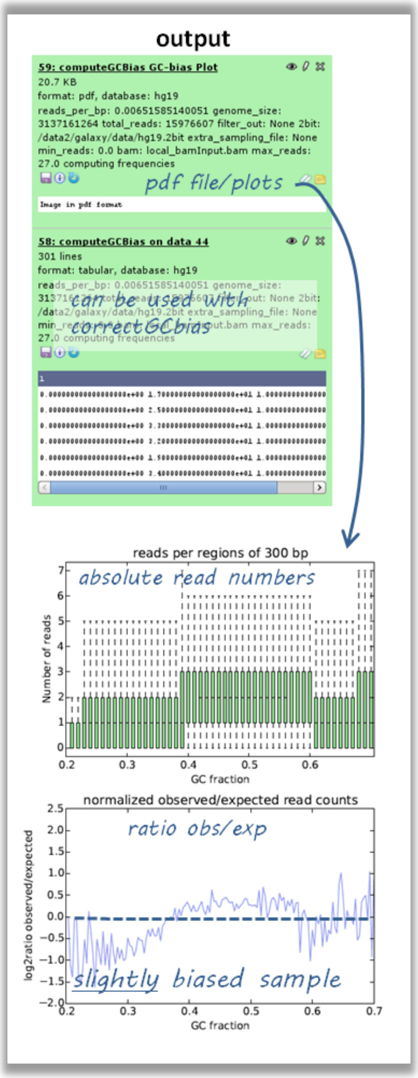
no

GC bias plot:

Image in pdf format

If given, a diagnostic image summarizing the GC bias found on the sample will be created.

Execute
select BAM file
select correct reference genome
indicate fragment length



output

59: computeGCBias GC-bias Plot

20.7 KB

format: pdf, database: hg19

reads_per_bp: 0.00651585140051 genome_size: 3137161264 total_reads: 15976607 filter_out: None 2bit: /data2/galaxy/data/hg19.2bit extra_sampling_file: None min_reads: 0.0 bam: local_bamInput.bam max_reads: 27.0 computing frequencies

Image in pdf format

58: computeGCBias on data 44

301 lines

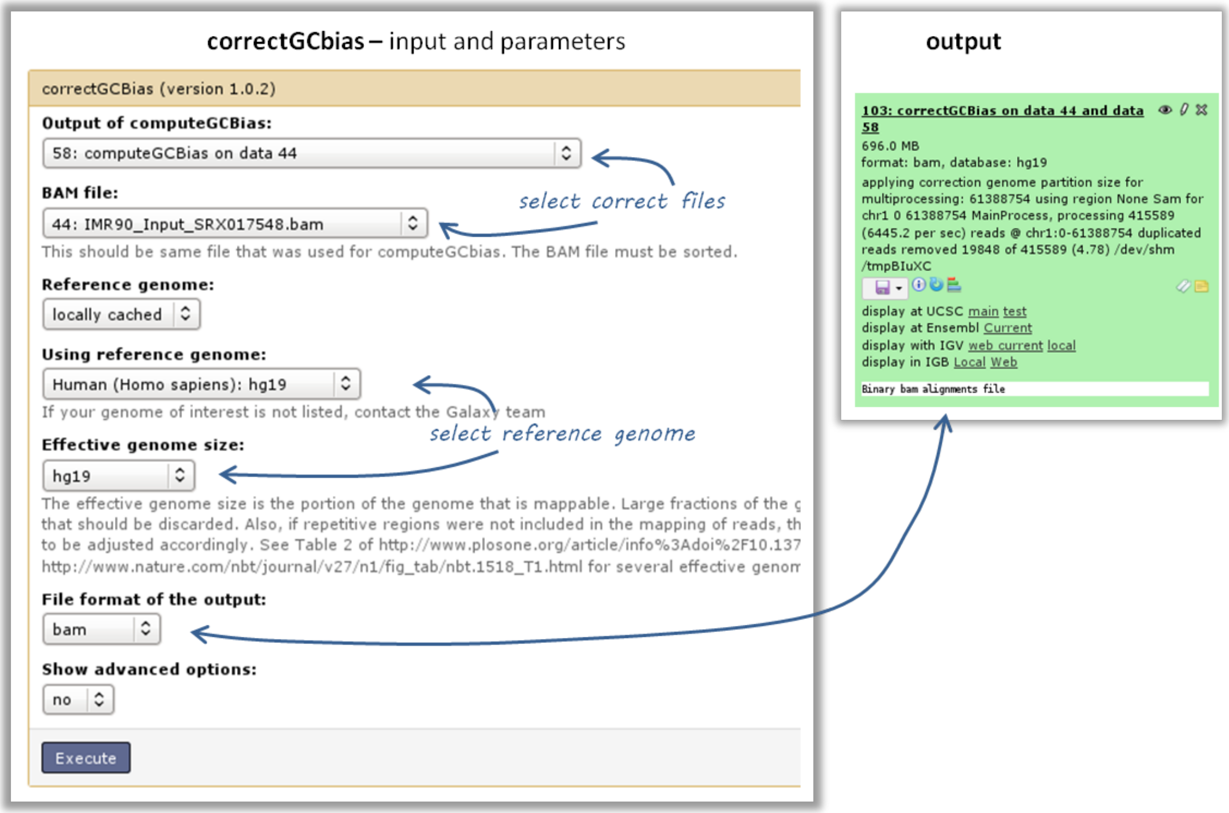
format: tabular, database: hg19

reads_per_bp: 0.00651585140051 genome_size: 3137161264 total_reads: 15976607 filter_out: None 2bit: /data2/galaxy/data/hg19.2bit extra_sampling_file: None min_reads: 0.0 bam: local_bamInput.bam max_reads: 27.0 computing frequencies

```
1
0.000000000000000000e+00 1.700000000000000000e+01 1.0000000000000
0.000000000000000000e+00 2.500000000000000000e+01 1.0000000000000
0.000000000000000000e+00 3.300000000000000000e+01 1.0000000000000
0.000000000000000000e+00 3.200000000000000000e+01 1.0000000000000
0.000000000000000000e+00 1.500000000000000000e+01 1.0000000000000
0.000000000000000000e+00 3.400000000000000000e+01 1.0000000000000
```

pdf file/plots
can be used with correctGCbias
reads per regions of 300 bp
absolute read numbers
Number of reads
0
1
2
3
4
5
6
7
0.2
0.3
0.4
0.5
0.6
GC fraction
normalized observed/expected read counts
ratio obs/exp
log2ratio observed/expected
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.2
0.3
0.4
0.5
0.6
0.7
GC fraction
slightly biased sample

- have a look at the image that is produced and compare it to the examples here
- if your sample shows an almost linear increase in exp/obs coverage (on the log scale of the lower plot), then you should consider correcting the GC bias - *if* you think that the biological interpretation of this data would otherwise be compromised (e.g. by comparing it to another sample that does not have an inherent GC bias)
  - the GC bias can be corrected with the tool tools/correctGCBias using the second output of the computeGCbias tool that you had to run anyway



correctGCBias (version 1.0.2)

Output of computeGCBias: 58: computeGCBias on data 44

BAM file: 44: IMR90_Input_SRX017548.bam

This should be same file that was used for computeGCbias. The BAM file must be sorted.

Reference genome: locally cached

Using reference genome: Human (Homo sapiens): hg19

If your genome of interest is not listed, contact the Galaxy team

Effective genome size: hg19

The effective genome size is the portion of the genome that is mappable. Large fractions of the g
that should be discarded. Also, if repetitive regions were not included in the mapping of reads, th
to be adjusted accordingly. See Table 2 of http://www.plosone.org/article/info%3Adoi%2F10.137
http://www.nature.com/nbt/journal/v27/n1/fig_tab/nbt.1518_T1.html for several effective genon

File format of the output: bam

Show advanced options: no

103: correctGCBias on data 44 and data 58

696.0 MB

format: bam, database: hg19

```
applying correction genome partition size for
multiprocessing: 61388754 using region None Sam for
chr1 0 61388754 MainProcess, processing 415589
(6445.2 per sec) reads @ chr1:0-61388754 duplicated
reads removed 19848 of 415589 (4.78) /dev/shm
/tmpBIuXC
```

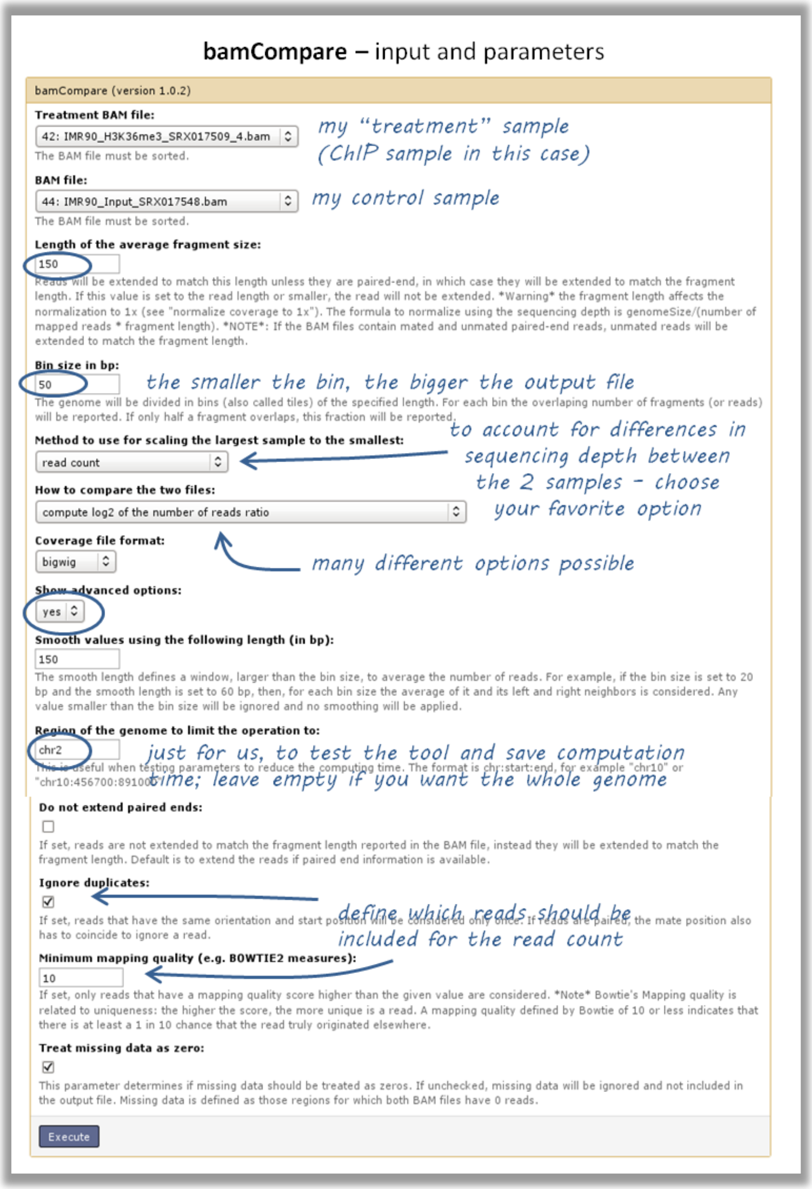
display at UCSC main test
display at Ensembl Current
display with IGV web current local
display in IGB Local Web

Binary bam alignments file

> **Warning:** `correctGCbias` will add reads to otherwise depleted regions (typically GC-poor regions), that means that you should **not** remove duplicates in any downstream analyses based on the GC-corrected BAM file. We therefore recommend removing duplicates before doing the correction so that only those duplicate reads are kept that were produced by the GC correction procedure.

How do I get an input-normalized ChIP-seq coverage file?

- input: you need two BAM files, one for the input and one for the ChIP-seq experiment
- tool: tools/bamCompare with ChIP = treatment, input = control sample



bamCompare – input and parameters

bamCompare (version 1.0.2)

Treatment BAM file:
42: IMR90_H3K36me3_SRX017509_4.bam
The BAM file must be sorted.

my "treatment" sample
(ChIP sample in this case)

BAM file:
44: IMR90_Input_SRX017548.bam
The BAM file must be sorted.

my control sample

Length of the average fragment size:
150
Reads will be extended to match this length unless they are paired-end, in which case they will be extended to match the fragment length. If this value is set to the read length or smaller, the read will not be extended. *Warning* the fragment length affects the normalization to 1x (see "normalize coverage to 1x"). The formula to normalize using the sequencing depth is genomeSize/(number of mapped reads * fragment length). *NOTE*: If the BAM files contain mated and unmated paired-end reads, unmated reads will be extended to match the fragment length.

Bin size in bp:
50

the smaller the bin, the bigger the output file

The genome will be divided in bins (also called tiles) of the specified length. For each bin the overlaping number of fragments (or reads) will be reported. If only half a fragment overlaps, this fraction will be reported.

Method to use for scaling the largest sample to the smallest:
read count

to account for differences in
sequencing depth between
the 2 samples - choose
your favorite option

How to compare the two files:
compute log2 of the number of reads ratio

Coverage file format:
bigwig

many different options possible

Show advanced options:
yes

Smooth values using the following length (in bp):
150
The smooth length defines a window, larger than the bin size, to average the number of reads. For example, if the bin size is set to 20 bp and the smooth length is set to 60 bp, then, for each bin size the average of it and its left and right neighbors is considered. Any value smaller than the bin size will be ignored and no smoothing will be applied.

Region of the genome to limit the operation to:
chr2
This is useful when testing parameters to reduce the computing time. The format is chr:start:end, for example "chr10" or "chr10:456700:891000".

just for us, to test the tool and save computation
time; leave empty if you want the whole genome

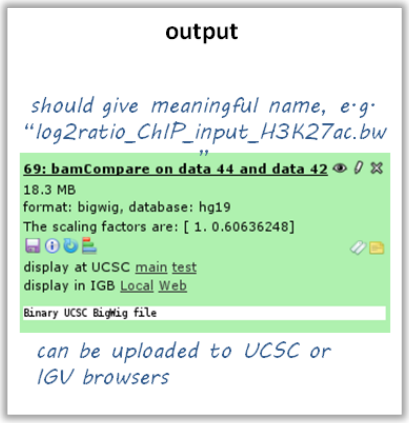
Do not extend paired ends:
☐
If set, reads are not extended to match the fragment length reported in the BAM file, instead they will be extended to match the fragment length. Default is to extend the reads if paired end information is available.

Ignore duplicates:
☑
If set, reads that have the same orientation and start position will be considered only once. If reads are paired, the mate position also has to coincide to ignore a read.

define which reads should be
included for the read count

Minimum mapping quality (e.g. BOWTIE2 measures):
10
If set, only reads that have a mapping quality score higher than the given value are considered. *Note* Bowtie's Mapping quality is related to uniqueness: the higher the score, the more unique is a read. A mapping quality defined by Bowtie of 10 or less indicates that there is at least a 1 in 10 chance that the read truly originated elsewhere.

Treat missing data as zero:
☑
This parameter determines if missing data should be treated as zeros. If unchecked, missing data will be ignored and not included in the output file. Missing data is defined as those regions for which both BAM files have 0 reads.



output

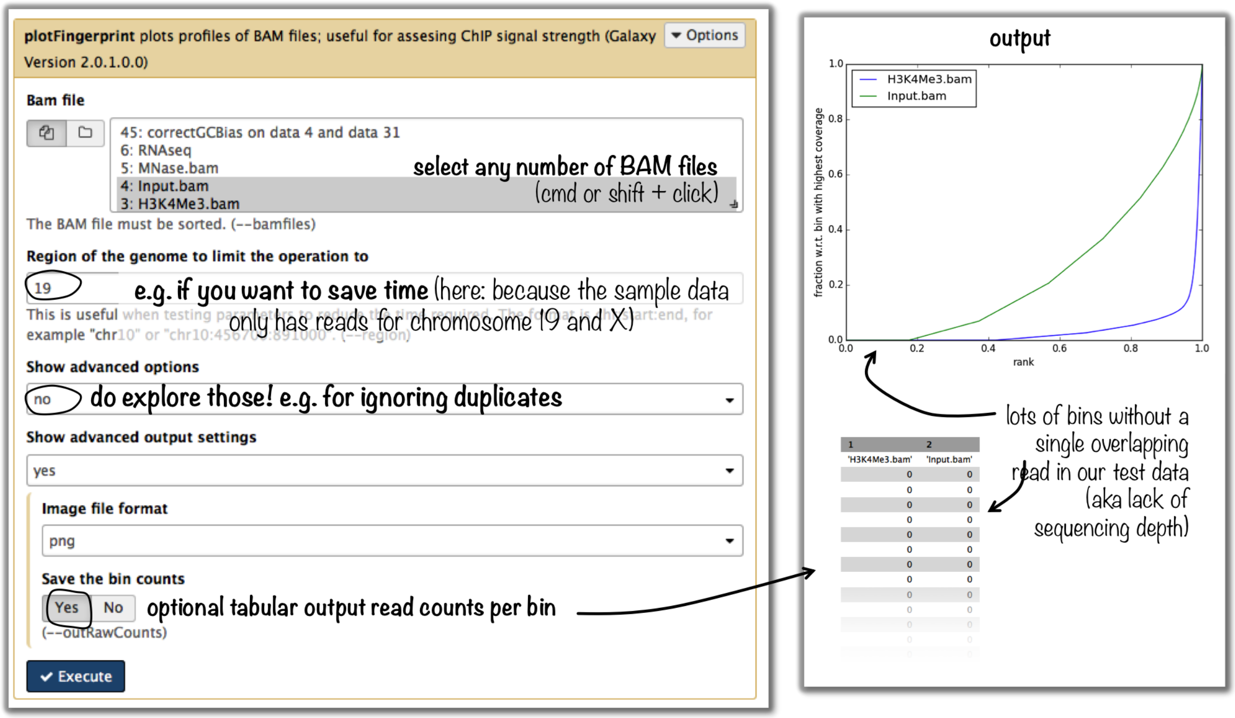
should give meaningful name, e.g.
"log2ratio_ChIP_input_H3K27ac.bw"

69: bamCompare on data 44 and data 42
18.3 MB
format: bigwig, database: hg19
The scaling factors are: [1. 0.60636248]
display at UCSC main test
display in IGB Local Web
Binary UCSC BigWig file

can be uploaded to UCSC or
IGV browsers

How can I compare the ChIP strength for different ChIP experiments?

- tool: tools/plotFingerprint
- input: as many BAM files of ChIP-seq samples as you'd like to compare (it is helpful to include the input control to see what a hopefully non-enriched sample looks like)



plotFingerprint plots profiles of BAM files; useful for assesing ChIP signal strength (Galaxy Version 2.0.1.0.0)

Bam file

45: correctGCBias on data 4 and data 31
6: RNAseq
5: MNase.bam
4: Input.bam
3: H3K4Me3.bam

select any number of BAM files (cmd or shift + click)

The BAM file must be sorted. (--bamfiles)

Region of the genome to limit the operation to: 19

e.g. if you want to save time (here: because the sample data only has reads for chromosome 19 and X)

This is useful when testing parameters to reduce the time required. The format is chr:start:end, for example "chr10" or "chr10:456700:891000". (--region)

Show advanced options: no

do explore those! e.g. for ignoring duplicates

Show advanced output settings: yes

Image file format: png

Save the bin counts: [✓ Yes] No

optional tabular output read counts per bin

(--outRawCounts)

output

lots of bins without a single overlapping read in our test data (aka lack of sequencing depth)

1	2
'H3K4Me3.bam'	'Input.bam'
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0

Tip: For more details on the interpretation of the plot, see tools/plotFingerprint or select the tool within the deepTools Galaxy and scroll down for more information.

Heatmaps and summary plots

How do I get a (clustered) heatmap of sequencing-depth-normalized read coverages around the transcription start site of all genes?

- tools: tools/computeMatrix, then tools/plotHeatmap
- **inputs:**
    - 1 *bigWig* file of normalized read coverages (e.g. the output of tools/bamCoverage or tools/bamCompare)
    - 1 *BED* or INTERVAL file of genes, e.g. obtained through Galaxy via "Get Data" –> "UCSC main table browser" –> group: "Genes and Gene Predictions" –> (e.g.) "RefSeqGenes" –> send to Galaxy (see screenshots below)

download gene lists

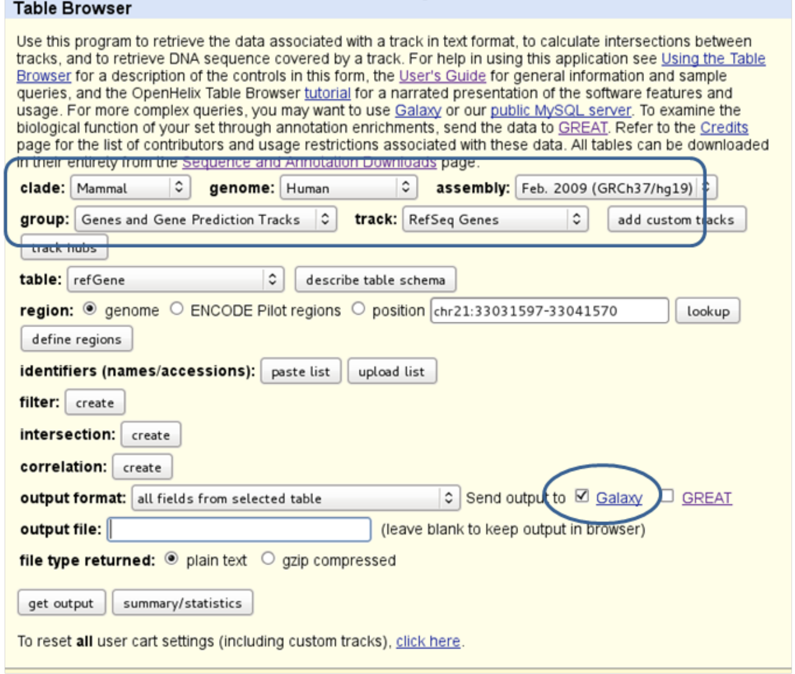


Table Browser
describe table schema
define regions
paste list
upload list
create
create
create
lookup
add custom tracks
track hubs
get output
summary/statistics

Use this program to retrieve the data associated with a track in text format, to calculate intersections between tracks, and to retrieve DNA sequence covered by a track. For help in using this application see Using the Table Browser for a description of the controls in this form, the User's Guide for general information and sample queries, and the OpenHelix Table Browser tutorial for a narrated presentation of the software features and usage. For more complex queries, you may want to use Galaxy or our public MySQL server. To examine the biological function of your set through annotation enrichments, send the data to GREAT. Refer to the Credits page for the list of contributors and usage restrictions associated with these data. All tables can be downloaded in their entirety from the Sequence and Annotation Downloads page.

clade: Mammal **genome:** Human **assembly:** Feb. 2009 (GRCh37/hg19)

group: Genes and Gene Prediction Tracks **track:** RefSeq Genes

table: refGene

region: ● genome ○ ENCODE Pilot regions ○ position chr21:33031597-33041570

identifiers (names/accessions):

filter:

intersection:

correlation:

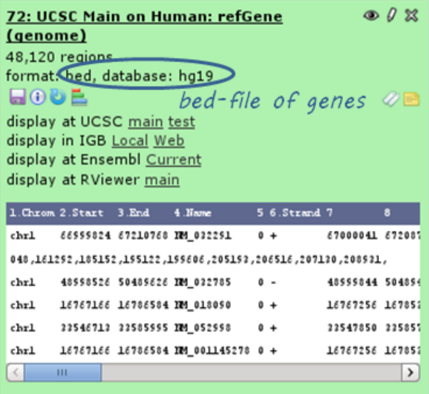
output format: all fields from selected table Send output to ☑ Galaxy ☐ GREAT

output file: (leave blank to keep output in browser)

file type returned: ● plain text ○ gzip compressed

To reset **all** user cart settings (including custom tracks), click here.

output


bed-file of genes
1.Chrom
2.Start
3.End
4.Name
5
6.Strand
7
8

72: UCSC Main on Human: refGene (genome)

48,120 regions

format: bed, database: hg19

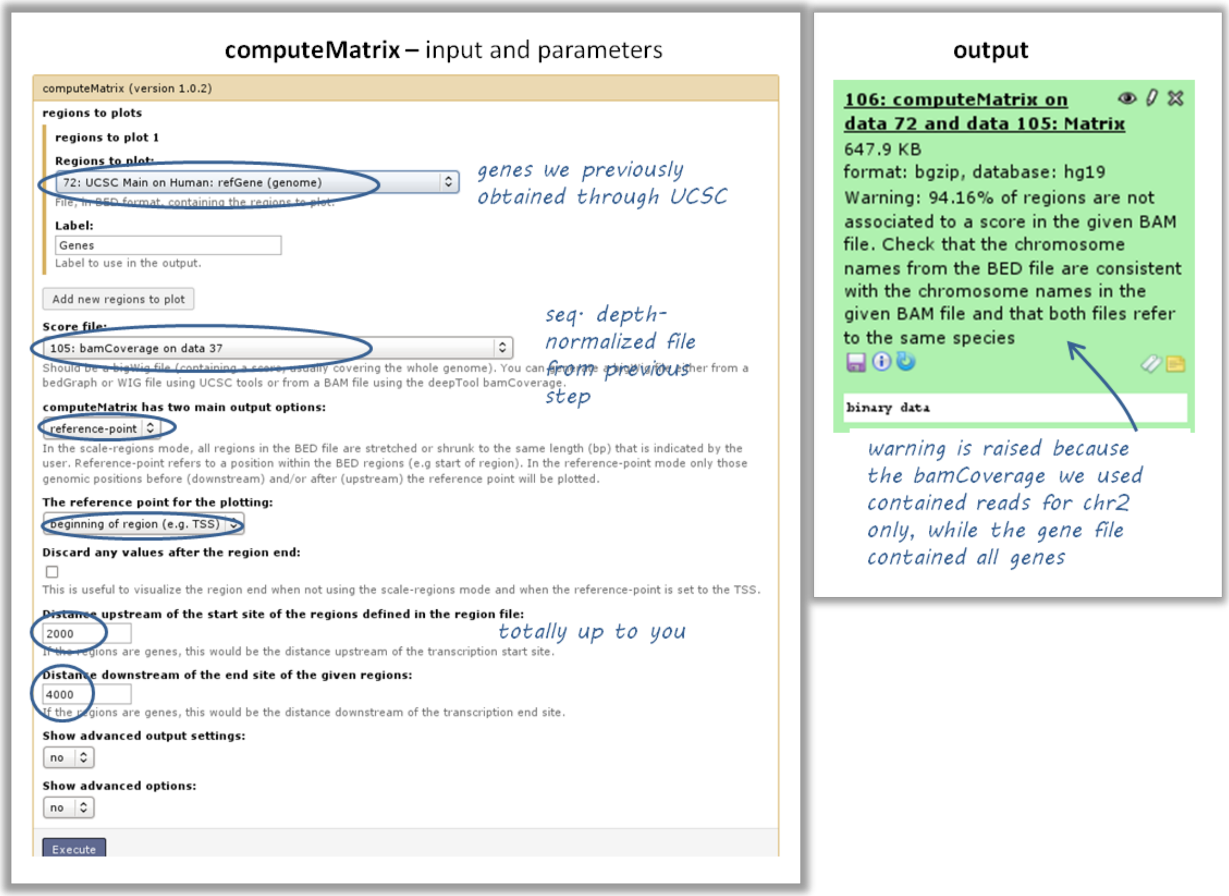
display at UCSC main test

display in IGB Local Web

display at Ensembl Current

display at RViewer main

- use tools/computeMatrix with the bigWig file and the BED file
- indicate `reference-point` (and whatever other option you would like to tune, see screenshot below)



computeMatrix – input and parameters

computeMatrix (version 1.0.2)

regions to plots

regions to plot 1

Regions to plot:
72: UCSC Main on Human: refGene (genome)
File, in BED format, containing the regions to plot.

genes we previously obtained through UCSC

Label:
Genes
Label to use in the output.

Add new regions to plot

Score file:
105: bamCoverage on data 37
Should be a bigWig file (containing a score, usually covering the whole genome). You can generate a bigWig file either from a bedGraph or WIG file using UCSC tools or from a BAM file using the deepTool bamCoverage.

seq. depth-normalized file from previous step

computeMatrix has two main output options:
reference-point
In the scale-regions mode, all regions in the BED file are stretched or shrunk to the same length (bp) that is indicated by the user. Reference-point refers to a position within the BED regions (e.g start of region). In the reference-point mode only those genomic positions before (downstream) and/or after (upstream) the reference point will be plotted.

The reference point for the plotting:
beginning of region (e.g. TSS)

Discard any values after the region end:
☐
This is useful to visualize the region end when not using the scale-regions mode and when the reference-point is set to the TSS.

Distance upstream of the start site of the regions defined in the region file:
2000
If the regions are genes, this would be the distance upstream of the transcription start site.

totally up to you

Distance downstream of the end site of the given regions:
4000
If the regions are genes, this would be the distance downstream of the transcription end site.

Show advanced output settings:
no

Show advanced options:
no

Execute

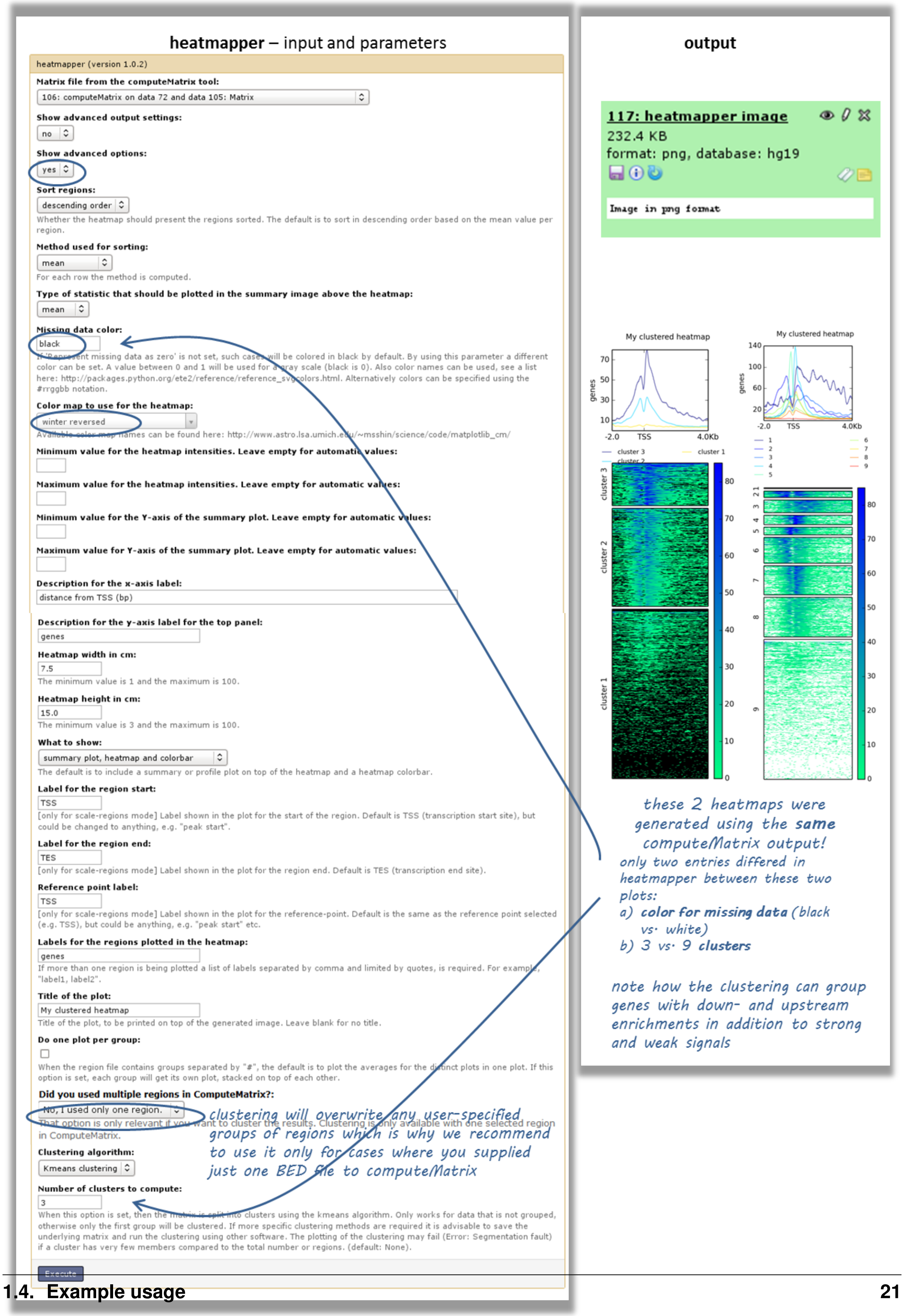
output

106: computeMatrix on data 72 and data 105: Matrix
647.9 KB
format: bgzip, database: hg19
Warning: 94.16% of regions are not associated to a score in the given BAM file. Check that the chromosome names from the BED file are consistent with the chromosome names in the given BAM file and that both files refer to the same species

binary data

warning is raised because the bamCoverage we used contained reads for chr2 only, while the gene file contained all genes

- **use the output from tools/computeMatrix with tools/plotHeatmap**
  - if you would like to cluster the signals, choose `k-means clustering` (last option of "advanced options") with a reasonable number of clusters (usually between 2 to 7)



heatmapper – input and parameters

heatmapper (version 1.0.2)

Matrix file from the computeMatrix tool:
106: computeMatrix on data 72 and data 105: Matrix

Show advanced output settings:
no

Show advanced options:
yes

Sort regions:
descending order

Whether the heatmap should present the regions sorted. The default is to sort in descending order based on the mean value per region.

Method used for sorting:
mean

For each row the method is computed.

Type of statistic that should be plotted in the summary image above the heatmap:
mean

Missing data color:
black

If 'Represent missing data as zero' is not set, such cases will be colored in black by default. By using this parameter a different color can be set. A value between 0 and 1 will be used for a gray scale (black is 0). Also color names can be used, see a list here: http://packages.python.org/ete2/reference/reference_svgcolors.html. Alternatively colors can be specified using the #rrggbb notation.

Color map to use for the heatmap:
winter reversed

Available color map names can be found here: http://www.astro.lsa.umich.edu/~msshin/science/code/matplotlib_cm/

Minimum value for the heatmap intensities. Leave empty for automatic values:

Maximum value for the heatmap intensities. Leave empty for automatic values:

Minimum value for the Y-axis of the summary plot. Leave empty for automatic values:

Maximum value for Y-axis of the summary plot. Leave empty for automatic values:

Description for the x-axis label:
distance from TSS (bp)

Description for the y-axis label for the top panel:
genes

Heatmap width in cm:
7.5

The minimum value is 1 and the maximum is 100.

Heatmap height in cm:
15.0

The minimum value is 3 and the maximum is 100.

What to show:
summary plot, heatmap and colorbar

The default is to include a summary or profile plot on top of the heatmap and a heatmap colorbar.

Label for the region start:
TSS

[only for scale-regions mode] Label shown in the plot for the start of the region. Default is TSS (transcription start site), but could be changed to anything, e.g. "peak start".

Label for the region end:
TES

[only for scale-regions mode] Label shown in the plot for the region end. Default is TES (transcription end site).

Reference point label:
TSS

[only for scale-regions mode] Label shown in the plot for the reference-point. Default is the same as the reference point selected (e.g. TSS), but could be anything, e.g. "peak start" etc.

Labels for the regions plotted in the heatmap:
genes

If more than one region is being plotted a list of labels separated by comma and limited by quotes, is required. For example, "label1, label2".

Title of the plot:
My clustered heatmap

Title of the plot, to be printed on top of the generated image. Leave blank for no title.

Do one plot per group:
☐

When the region file contains groups separated by "#", the default is to plot the averages for the distinct plots in one plot. If this option is set, each group will get its own plot, stacked on top of each other.

Did you used multiple regions in ComputeMatrix?:
No, I used only one region.

That option is only relevant if you want to cluster the results. Clustering is only available with one selected region in ComputeMatrix.

Clustering algorithm:
Kmeans clustering

Number of clusters to compute:
3

When this option is set, then the matrix is split into clusters using the kmeans algorithm. Only works for data that is not grouped, otherwise only the first group will be clustered. If more specific clustering methods are required it is advisable to save the underlying matrix and run the clustering using other software. The plotting of the clustering may fail (Error: Segmentation fault) if a cluster has very few members compared to the total number or regions. (default: None).

clustering will overwrite any user-specified groups of regions which is why we recommend to use it only for cases where you supplied just one BED file to computeMatrix

output

117: heatmapper image
232.4 KB
format: png, database: hg19

Image in png format

these 2 heatmaps were generated using the **same** computeMatrix output!

only two entries differed in heatmapper between these two plots:

a) **color for missing data** (black vs. white)

b) 3 vs. 9 **clusters**

note how the clustering can group genes with down- and upstream enrichments in addition to strong and weak signals

---

How can I compare the average signal for X-specific and autosomal genes for 2 or more different sequencing experiments?

Make sure you're familiar with computeMatrix and plotProfile before using this protocol.

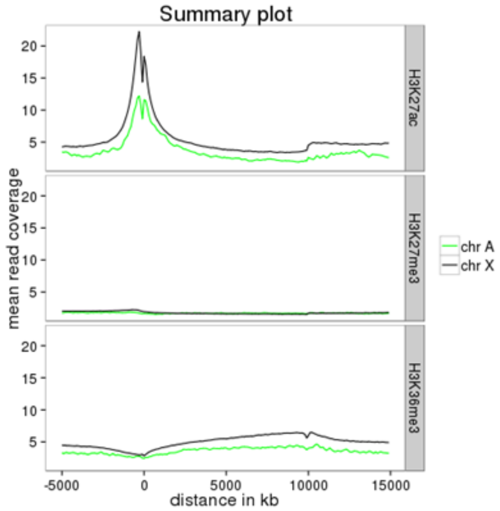
- **tools:**
    - Filter data on any column using simple expressions
    - computeMatrix
    - plotProfile
    - (plotting the summary plots for multiple samples)
- **inputs:**
    - several bigWig files (one for each sequencing experiment you would like to compare)
    - two BED files, one with X-chromosomal and one with autosomal genes

How to obtain a BED file for X chromosomal and autosomal genes each

1. download a full list of genes via "Get Data" –> "UCSC main table browser" –> group:"Genes and Gene Predictions" –> tracks: (e.g.) "RefSeqGenes" –> send to Galaxy
2. filter the list twice using the tool **"Filter data on any column using simple expressions"**
    - first use the expression: c1=="chrX" to filter the list of all genes –> this will generate a list of X-linked genes
    - then re-run the filtering, now with c1!="chrX", which will generate a list of genes that do not belong to chromosome X (!= indicates "not matching")

Compute the average values for X and autosomal genes

- use tools/computeMatrix for all of the signal files (bigWig format) at once
    - supply both filtered BED files (click on "Add new regions to plot" once) and label them
    - indicate the corresponding signal files
- now use tools/plotProfile on the resulting file
    - important: display the "advanced output options" and select "save the data underlying the average profile" –> this will generate a table in addition to the summary plot images



deepTools Galaxy.	code @ github.

1.4.2 Gallery of deepTools plots

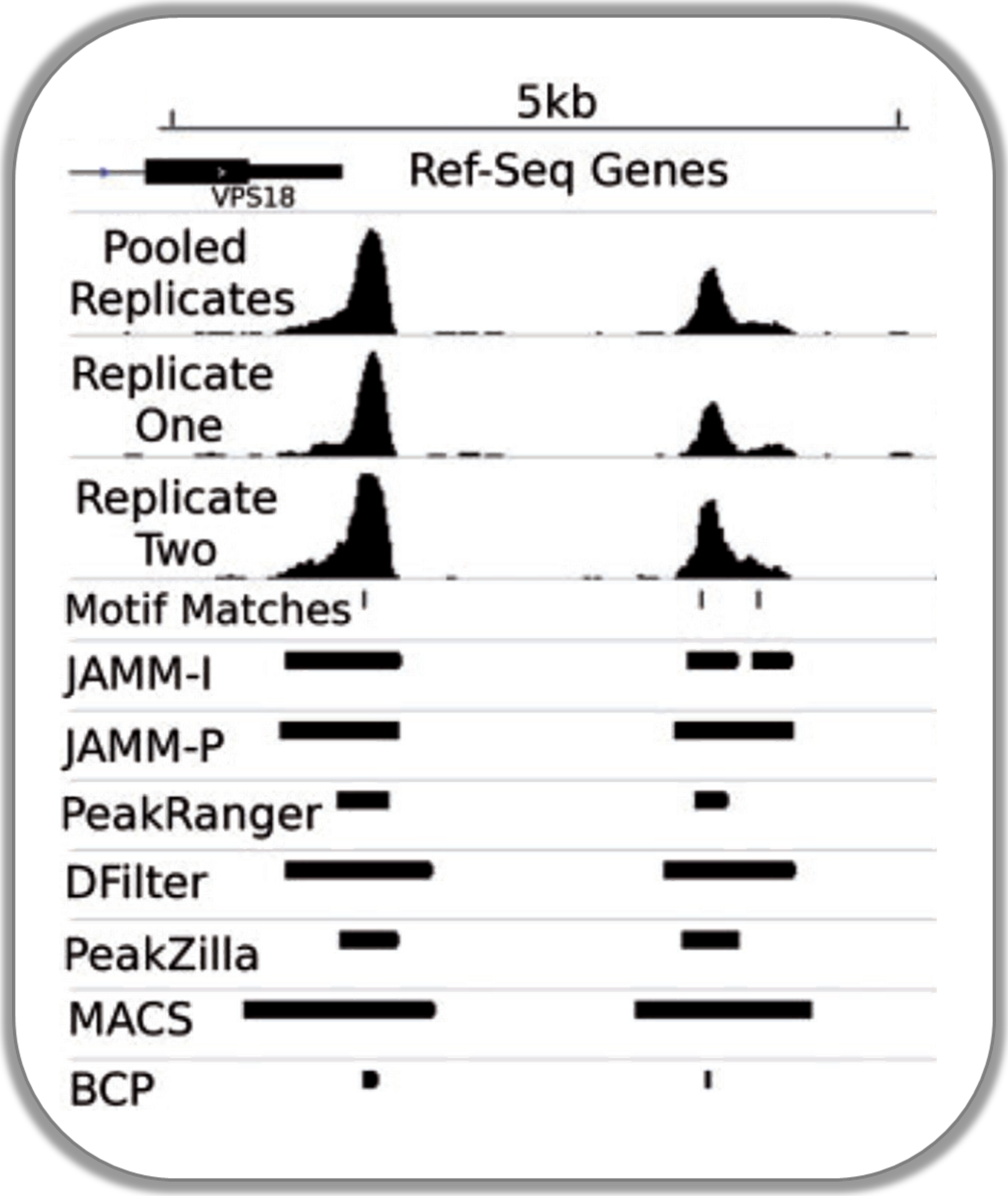
Published example plots

- *Normalized ChIP-seq signals and peak regions*
- *DNase accessibility at enhancers in murine ES cells*
- *TATA box enrichments around the TSS of mouse genes*
- *Visualizing the GC content for mouse and fly genes*
- *CpG methylation around murine transcription start sites in two different cell types*
- *Histone marks for genes of the mosquito* Anopheles gambiae
- *Signals of repressive chromatin marks, their enzymes and repeat element conservation scores*

We're trying to collect a wide variety of plots generated using deepTools. For the plots that we created ourselves, we try to point out the options that were used to create each image, so perhaps these can serve as inspiration for you.

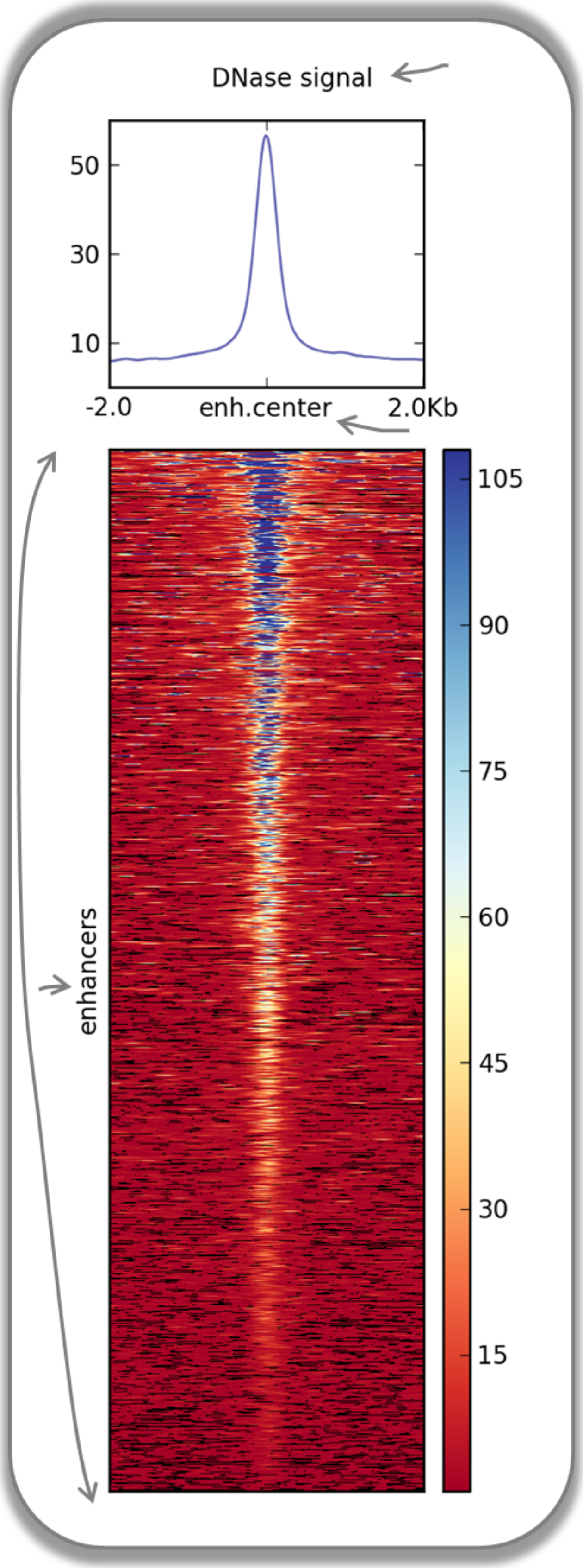
Normalized ChIP-seq signals and peak regions

This image was published by Ibrahim et al., 2014 (NAR). They used deepTools to generate extended reads per kilobase per million reads at 10 base resolution and visualized the resulting coverage files in IGV.



DNase accessibility at enhancers in murine ES cells

The following image demonstrates that enhancer regions are typically small stretches of highly accessible chromatin (more information on enhancers can be found, for example, here). In the heatmap, yellow and blue tiles indicate a large numbers of reads that were sequenced (indicative of open chromatin) and black spots indicate missing data points. An appropriate labeling of the y-axis was neglected.


DNase signal
50
30
10
-2.0
enh.center
2.0Kb
105
90
75
60
45
30
15
enhancers

Fast Facts:

- *computeMatrix* mode: reference-point
- *regions file*: BED file with typical enhancer regions from Whyte et al., 2013 (download here)
- *signal file*: bigWig file with DNase signal from UCSC
- *heatmap cosmetics*: labels, titles, heatmap height

Command:

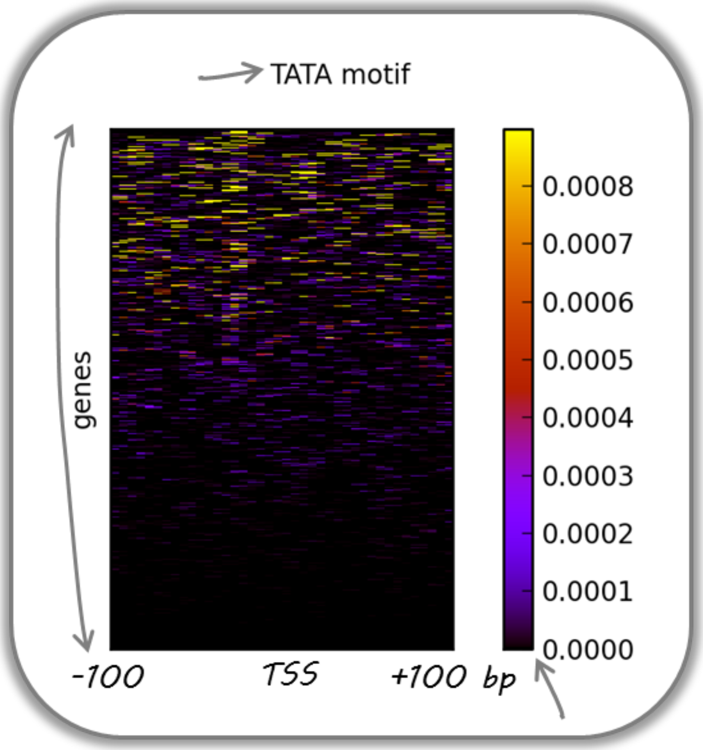
```
$ computeMatrix reference-point \
 -S DNase_mouse.bigwig \
 -R Whyte_TypicalEnhancers_ESC.bed \
 --referencePoint center \
 -a 2000 -b 2000 \ ## regions before and after the enhancer centers
 -out matrix_Enhancers_DNase_ESC.tab.gz

$ plotHeatmap \
 -m matrix_Enhancers_DNase_ESC.tab.gz\
 -out hm_DNase_ESC.png \
 --heatmapHeight 15  \
 --refPointLabel enh.center \
 --regionsLabel enhancers \
 --plotTitle 'DNase signal' \
```

TATA box enrichments around the TSS of mouse genes

Using the TRAP suite, we produced a bigWig file that contained TRAP scores for the well-known TATA box motif along the mouse genome. The TRAP score is a measure for the strength of a protein-DNA interaction at a given DNA sequence; the higher the score, the closer the motif is to the consensus motif sequence. The following heatmap demonstrates that:

- TATA-like motifs occur quite frequently
- there is an obvious clustering of TATA motifs slightly upstream of the TSS of many mouse genes
- there are many genes that do not contain TATA-like motifs at their promoter

Note that the heatmap shows *all* mouse RefSeq genes, so ca. 15,000 genes!



Fast Facts:

- *computeMatrix mode*: reference-point
- *regions file*: BED file with all mouse genes (from UCSC table browser)
- *signal file*: bigWig file of TATA psem scores
- *heatmap cosmetics*: color scheme, labels, titles, heatmap height, only showing heatmap + colorbar

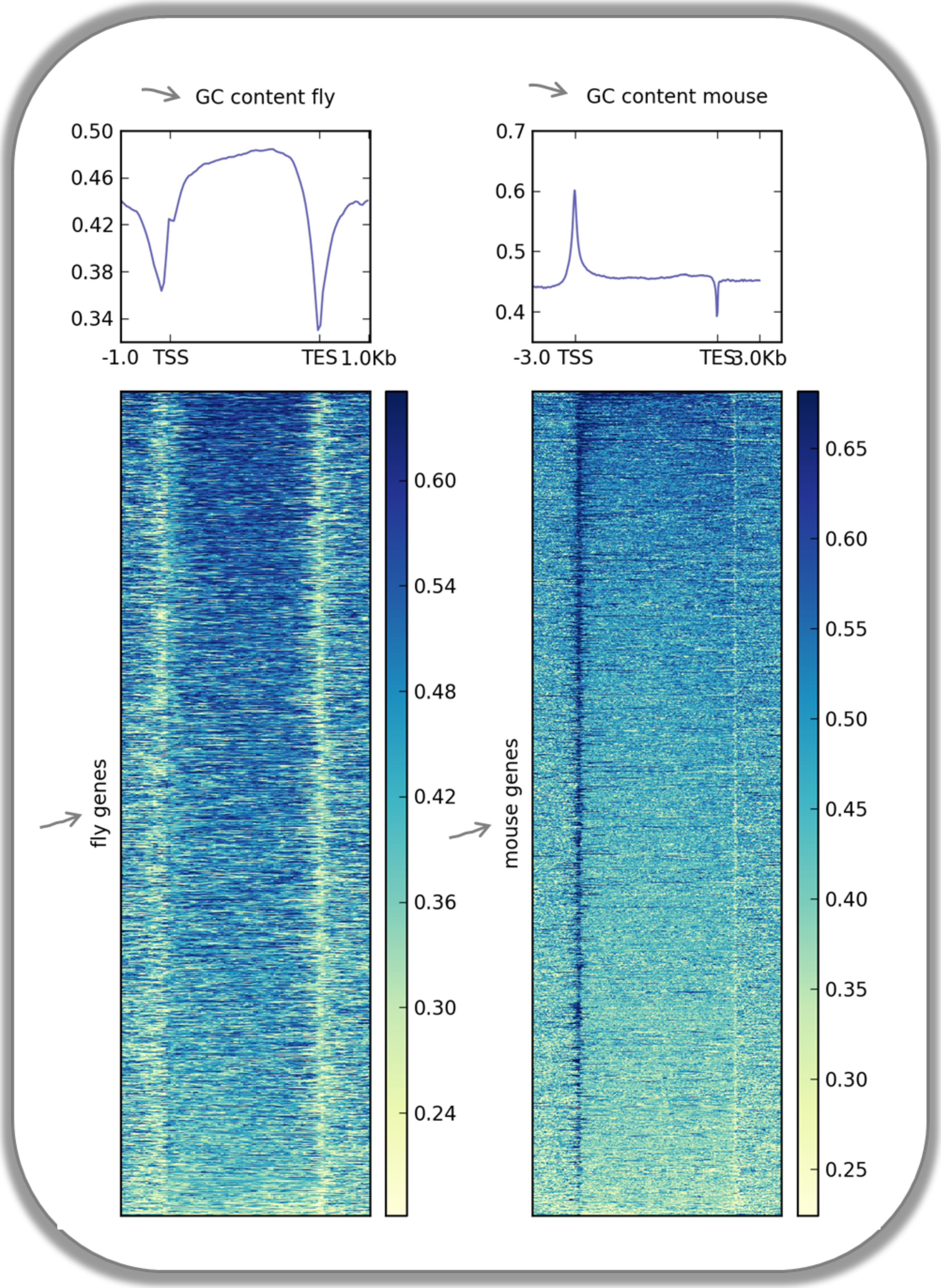
Command:

```
$ computeMatrix reference-point \
 -S TATA_01_pssm.bw \
 -R RefSeq_genes.bed \
 --referencePoint TSS \
 -a 100 -b 100 \
 --binSize 5 \

$ plotHeatmap \
 -m matrix_Genes_TATA.tab.gz  \
 -out hm_allGenes_TATA.png \
 --colorMap hot_r \
 --missingDataColor .4 \
 --heatmapHeight 7 \
 --plotTitle 'TATA motif' \
 --whatToShow 'heatmap and colorbar' \
 --sortRegions ascend
```

Visualizing the GC content for mouse and fly genes

It is well known that different species have different genome GC contents. Here, we used two bigWig files where the GC content was calculated for 50 base windows along the genome of mice and flies and the resulting scores visualized for gene regions.

The images nicely illustrate the completely opposite GC distributions in flies and mice: while the gene starts of mammalian genomes are enriched for Gs and Cs, fly promoters show depletion of GC content.


GC content fly
0.50
0.46
0.42
0.38
0.34
-1.0
TSS
TES
1.0Kb
fly genes
0.60
0.54
0.48
0.42
0.36
0.30
0.24
GC content mouse
0.7
0.6
0.5
0.4
-3.0
TSS
TES
3.0Kb
mouse genes
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25

Fast Facts	
computeMatrix mode	scale-regions
regions files	BED files with mouse and fly genes (from UCSC table browser)
signal file	bigwig files with GC content
heatmap cosmetics	color scheme, labels, titles, color for missing data was set to white, heatmap height

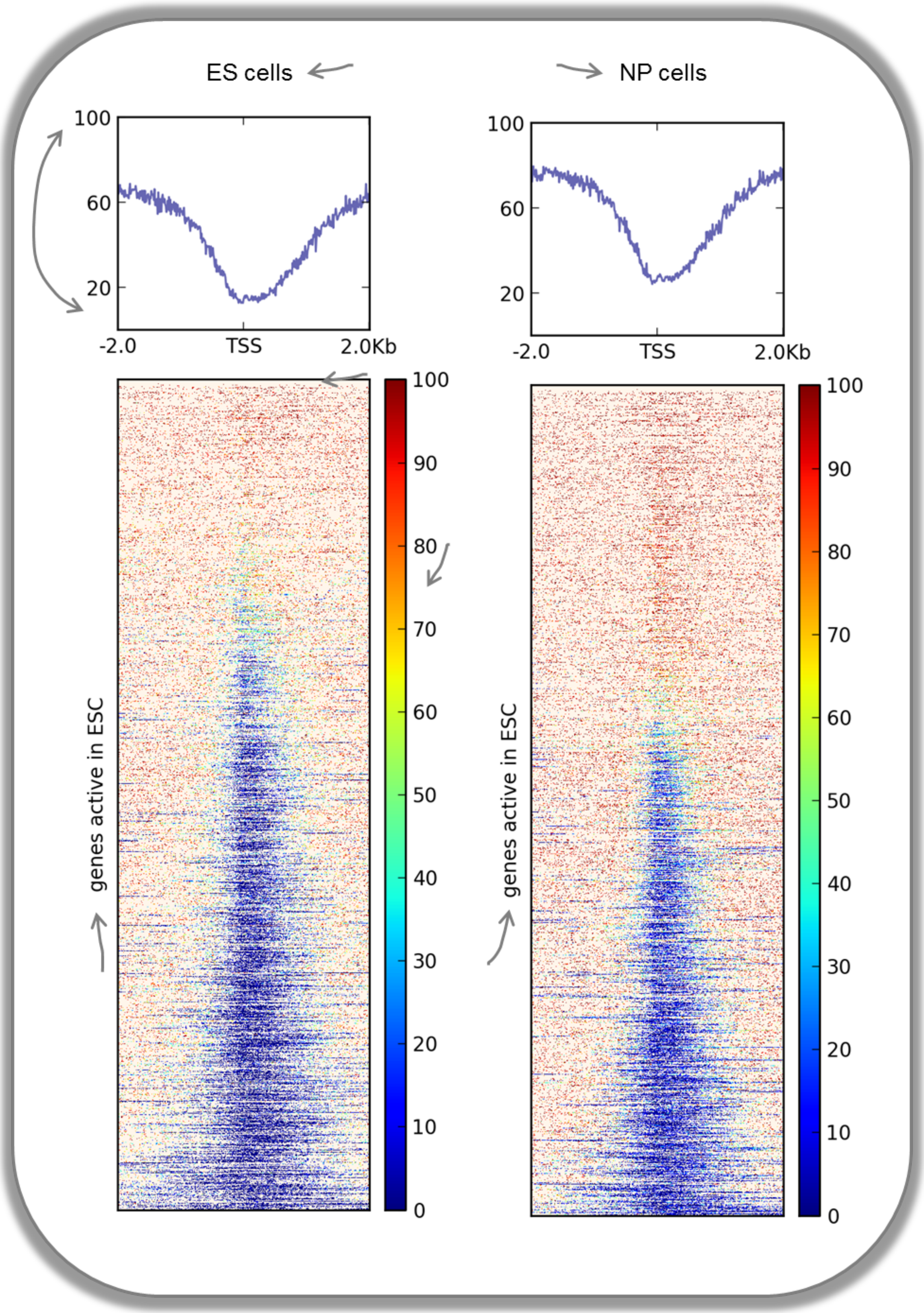
Fly and mouse genes were scaled to different sizes due to the different median sizes of the two species' genes (genes of *D.melanogaster* contain many fewer introns and are considerably shorter than mammalian genes). Thus, computeMatrix had to be run with slightly different parameters while the plotHeatmap commands were virtually identical (except for the labels).

```
$ computeMatrix scale-regions \
 -S GCcontent_Mm9_50_5.bw \
 -R RefSeq_genes_uniqNM.bed \
 -bs 50
 -m 10000 -b 3000 -a 3000 \
 -out matrix_GCcont_Mm9_scaledGenes.tab.gz \
 --skipZeros \
 --missingDataAsZero

$ computeMatrix scale-regions \
 -S GCcontent_Dm3_50_5.bw \
 -R Dm530.genes.bed \
 -bs 50
 -m 3000 -b 1000 -a 1000 \
 -out matrix_GCcont_Dm3_scaledGenes.tab.gz \
 --skipZeros --missingDataAsZero

$ plotHeatmap \
 -m matrix_GCcont_Dm3_scaledGenes.tab.gz \
 -out hm_GCcont_Dm3_scaledGenes.png \
 --colorMap YlGnBu \
 --regionsLabel 'fly genes' \
 --heatmapHeight 15 \
 --plotTitle 'GC content fly' &

$ plotHeatmap \
 -m matrix_GCcont_Mm9_scaledGenes.tab.gz \
 -out hm_GCcont_Mm9_scaledGenes.png \
 --colorMap YlGnBu \
 --regionsLabel 'mouse genes' \
 --heatmapHeight 15 \
 --plotTitle 'GC content mouse' &
```

CpG methylation around murine transcription start sites in two different cell types

In addition to the methylation of histone tails, the cytosines can also be methylated (for more information on CpG methylation, read here). In mammalian genomes, most CpGs are methylated unless they are in gene promoters that need to be kept unmethylated to allow full transcriptional activity. In the following heatmaps, we used genes expressed primarily in ES cells and checked the percentages of methylated cytosines around their transcription start sites. The blue signal indicates that very few methylated cytosines are found. When you compare the CpG methylation signal between ES cells and neuronal progenitor (NP) cells, you can see that the majority of genes remain unmethylated, but the general amount of CpG methylation around the TSS increases, as indicated by the stronger red signal and the slight elevation of the CpG methylation signal in the summary plot. This supports the notion that genes stored in the BED file indeed tend to be more expressed in ES than in NP cells.

This image was taken from Chelmicki & Dündar et al. (2014), eLife.


ES cells
NP cells
100
60
20
-2.0
TSS
2.0Kb
100
90
80
70
60
50
40
30
20
10
0
genes active in ESC

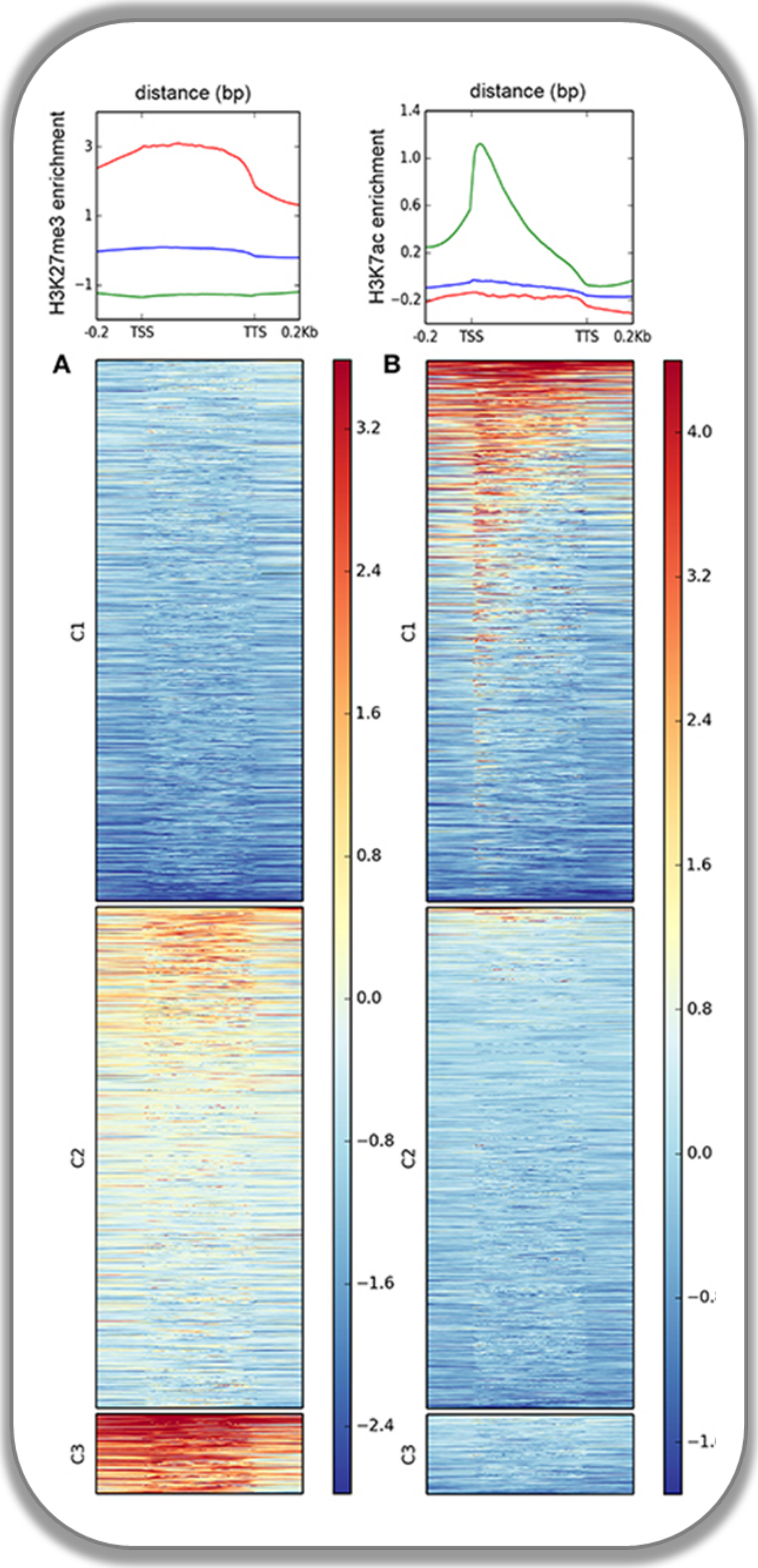
Fast Facts	
computeMatrix mode	reference-point
regions files	*BED* file mouse genes expressed in ES cells
signal file	*bigWig* files with fraction of methylated cytosins (from Stadler et al., 2011)
heatmap cosmetics	color scheme, labels, titles, color for missing data was set to customized color, y-axis of profiles were changed, heatmap height

The commands for the bigWig files from the ES and NP cells were the same:

```
$ computeMatrix reference-point \
 -S GSE30202_ES_CpGmeth.bw \
 -R activeGenes_ESConly.bed \
 --referencePoint TSS \
 -a 2000 -b 2000 \
 -out matrix_Genes_ES_CpGmeth.tab.gz

$ plotHeatmap \
 -m matrix_Genes_ES_CpGmeth.tab.gz \
 -out hm_activeESCGenes_CpG_ES_indSort.png \
 --colorMap jet \
 --missingDataColor "#FFF6EB" \
 --heatmapHeight 15 \
 --yMin 0 --yMax 100 \
 --plotTitle 'ES cells' \
 --regionsLabel 'genes active in ESC'
```

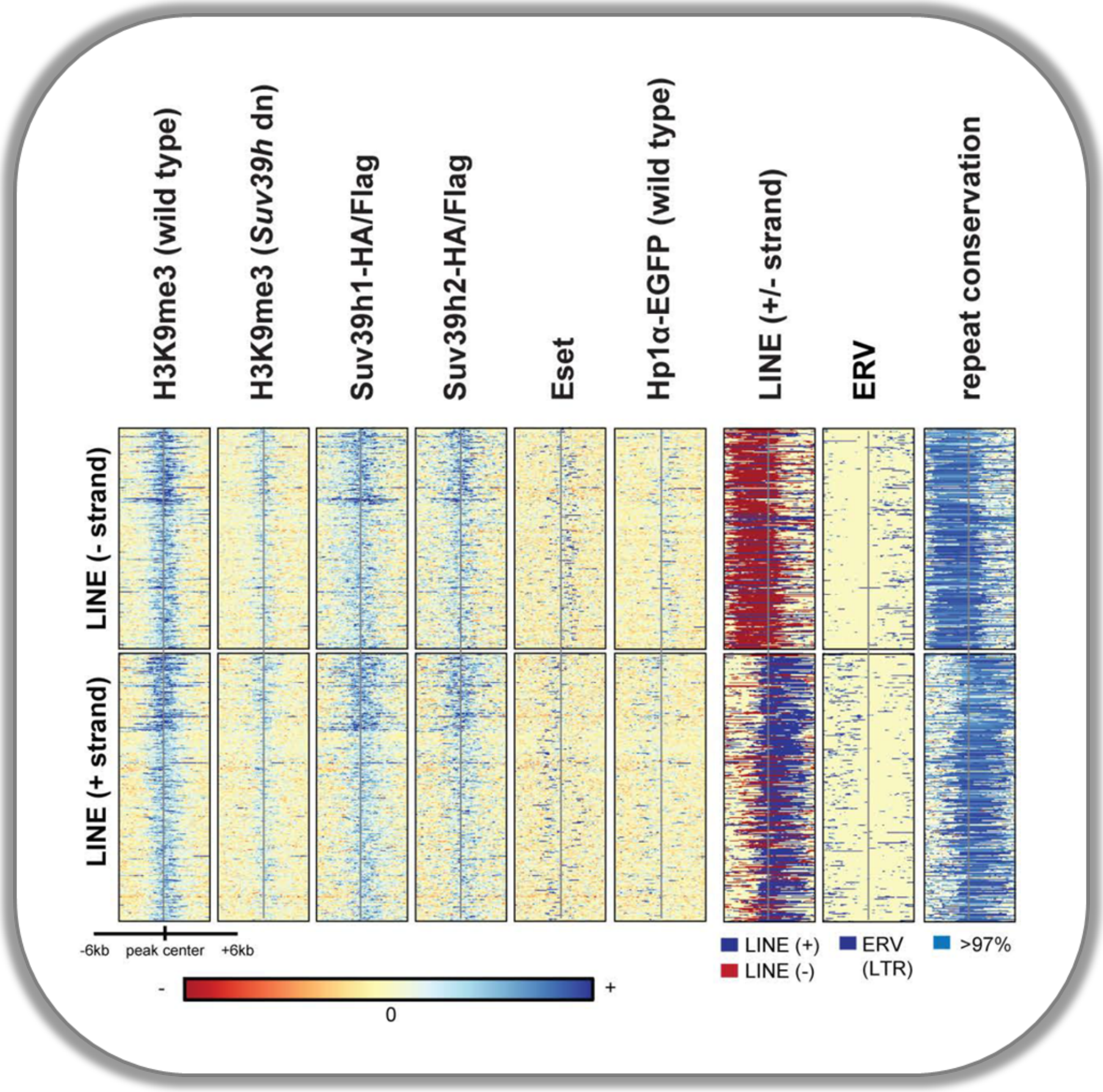
Histone marks for genes of the mosquito *Anopheles gambiae*

This figure was taken from Gómez-Díaz et al. (2014): Insights into the epigenomic landscape of the human malaria vector *Anopheles gambiae*. From Genet Aug15;5:277. It shows the distribution of H3K27Me3 (left) and H3K27Ac (right) over gene features in *A. gambiae* midguts. The enrichment or depletion is shown relative to chromatin input. The regions in the map comprise gene bodies flanked by a segment of 200 bases at the 5 end of TSSs and TTSs. Average profile across gene regions ±200 bases for each histone modification are shown on top.


distance (bp)
H3K27me3 enrichment
3
1
−1
-0.2
TSS
TTS
0.2Kb
distance (bp)
H3K7ac enrichment
1.4
1.0
0.6
0.2
−0.2
-0.2
TSS
TTS
0.2Kb
A
C1
C2
C3
3.2
2.4
1.6
0.8
0.0
−0.8
−1.6
−2.4
B
C1
C2
C3
4.0
3.2
2.4
1.6
0.8
0.0

Signals of repressive chromatin marks, their enzymes and repeat element conservation scores

This image is from Bulut-Karsliogu and De La Rosa-Velázquez et al. (2014), Mol Cell. The heatmaps depict various signal types for unscaled peak regions of proteins and histone marks associated with repressed chromatin. The peaks were separated into those containing long interspersed elements (LINEs) on the forward and reverse strand. The signals include normalized ChIP-seq signals for H3K9Me3, Suv39h1, Suv39h2, Eset, and HP1alpha-EGFP, followed by LINE and ERV content and repeat conservation scores.

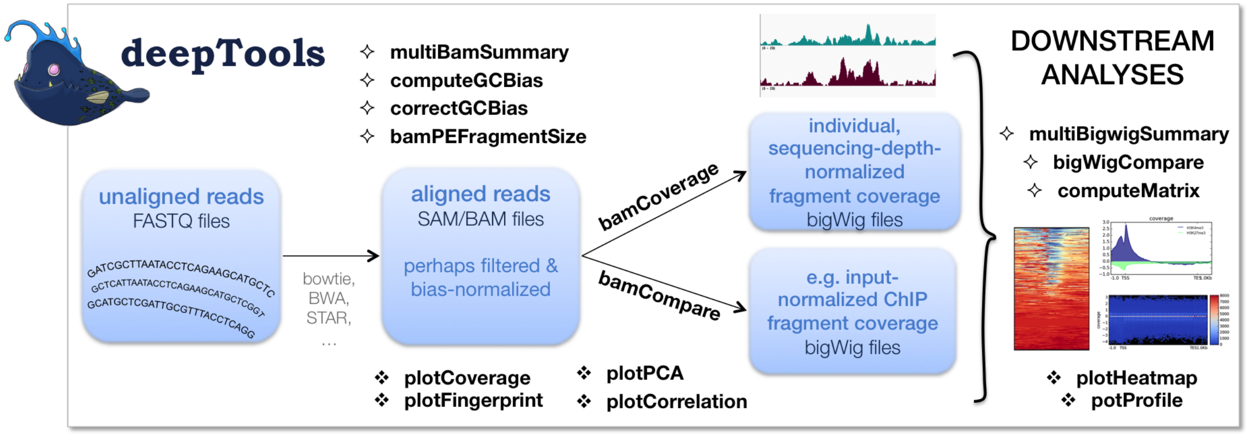


deepTools Galaxy. | code @ github.

1.4.3 How we use deepTools for ChIP-seq analyses

To get a feeling for what deepTools can do, we'd like to give you a brief glimpse into how we typically use deepTools for ChIP-seq analyses. For more detailed exampes and descriptions of the tools, simply follow the respective links.

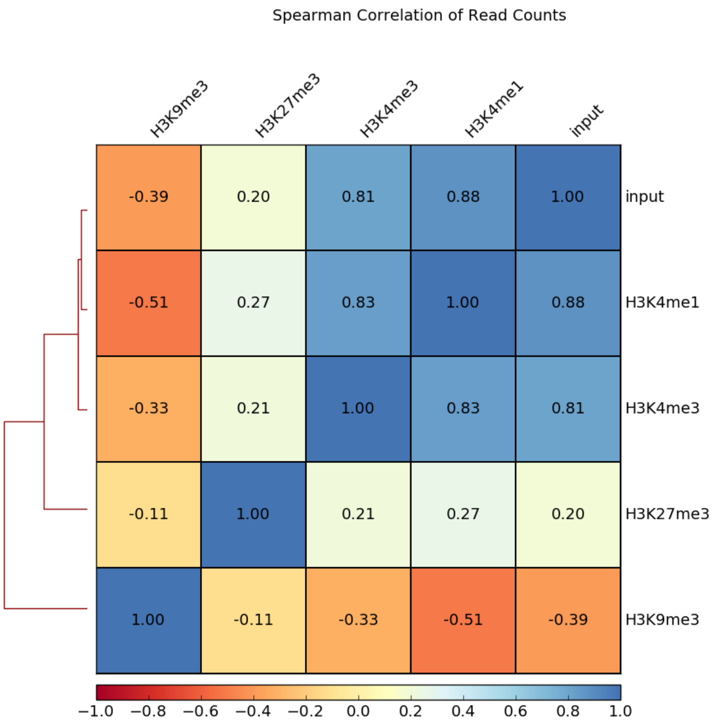
Note: While some tools, such as tools/plotFingerprint, specifically address ChIP-seq-issues, the majority of tools is widely applicable to deep-sequencing data, including RNA-seq.


deepTools
✧ multiBamSummary
✧ computeGCBias
✧ correctGCBias
✧ bamPEFragmentSize
unaligned reads
FASTQ files
GATCGCTTAATACCTCAGAAGCATGCTC
GCTCATTAATACCTCAGAAGCATGCTCGGT
GCATGCTCGATTGCGTTTACCTCAGG
bowtie,
BWA,
STAR,
…
aligned reads
SAM/BAM files
perhaps filtered &
bias-normalized
bamCoverage
bamCompare
individual,
sequencing-depth-
normalized
fragment coverage
bigWig files
e.g. input-
normalized ChIP
fragment coverage
bigWig files
DOWNSTREAM
ANALYSES
✧ multiBigwigSummary
✧ bigWigCompare
✧ computeMatrix
✣ plotCoverage
✣ plotFingerprint
✣ plotPCA
✣ plotCorrelation
✣ plotHeatmap
✣ potProfile

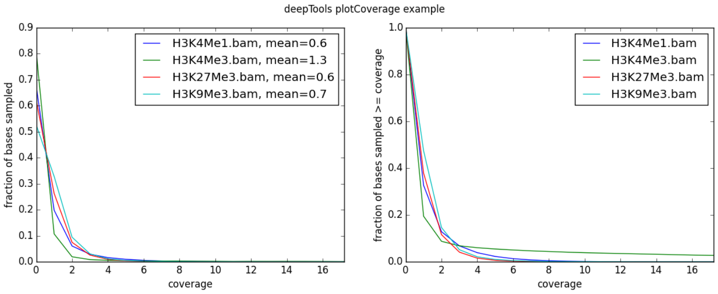
As shown in the flow chart above, our work usually begins with one or more *FASTQ* file(s) of deeply-sequenced samples. After preliminary quality control using FASTQC, we align the reads to the reference genome, e.g., using bowtie2. The standard output of bowtie2 (and other mapping tools) is in the form of sorted and indexed *BAM* files that provide the common input and starting point for all subsequent deepTools analyses. We then use deepTools to assess the quality of the aligned reads:

1. **Correlation between BAM files** (tools/multiBamSummary and tools/plotCorrelation). Together, these two modules perform a very basic test to see whether the sequenced and aligned reads meet your expectations. We use this check to assess reproducibility - either between replicates and/or between different experiments that might have used the same antibody or the same cell type, etc. For instance, replicates should correlate better than differently treated samples.

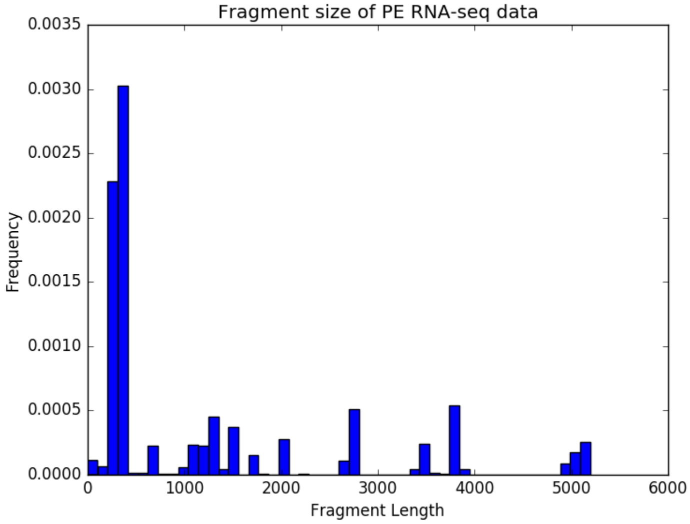
   Tip: You can also assess the correlation of *bigWig* files using tools/multiBigwigSummary.



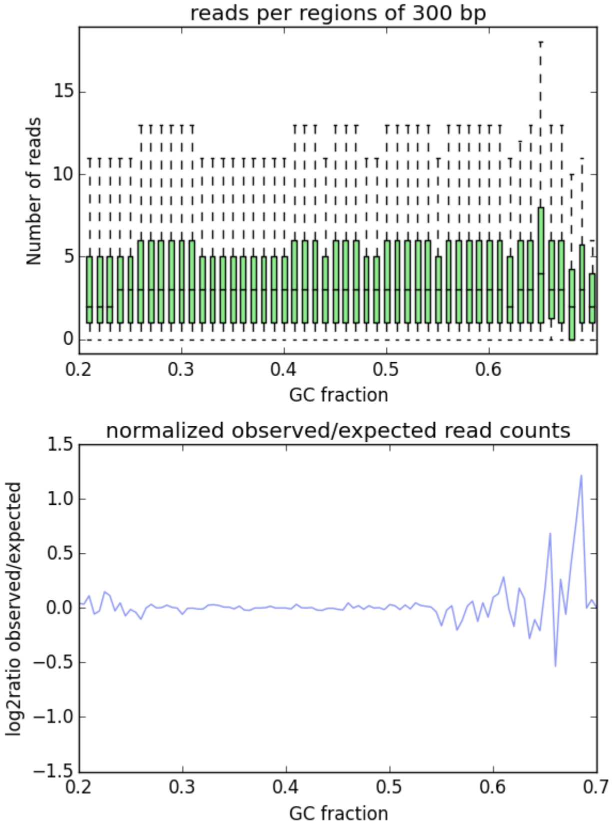
2. **Coverage check** (tools/plotCoverage). To see how many bp in the genome are actually covered by (a good number) of sequencing reads, we use tools/plotCoverage which generates two diagnostic plots that help us decide whether we need to sequence deeper or not. The option `--ignoreDuplicates` is particularly useful here!



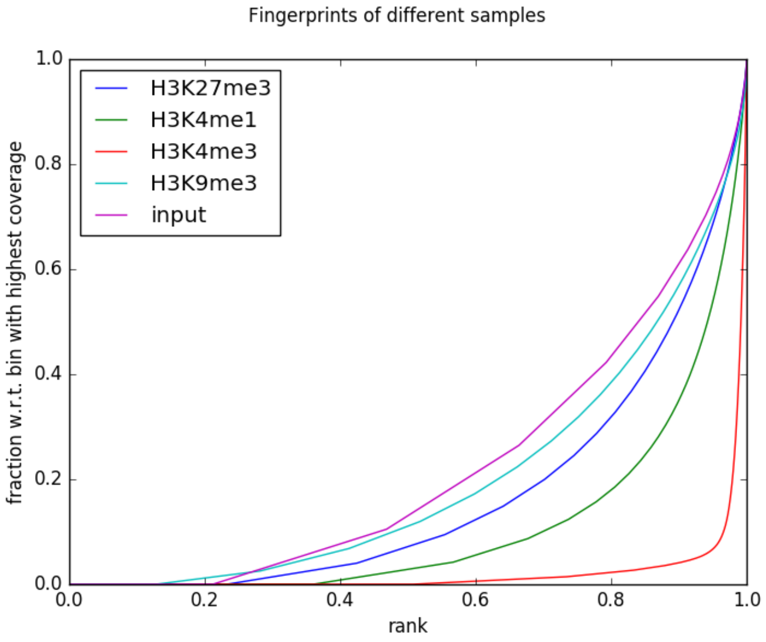
For paired-end samples, we often additionally check whether the fragment sizes are more or less what we would expected based on the library preparation. The module tools/bamPEFragmentSize can be used for that.



3. **GC-bias check** (tools/computeGCBias). Many sequencing protocols require several rounds of PCR-based DNA amplification, which often introduces notable bias, due to many DNA polymerases preferentially amplifying GC-rich templates. Depending on the sample (preparation), the GC-bias can vary significantly and we routinely check its extent. When we need to compare files with different GC biases, we use the tools/correctGCBias module. See the paper by Benjamini and Speed for many insights into this problem.



4. **Assessing the ChIP strength**. We do this quality control step to get a feeling for the signal-to-noise ratio in samples from ChIP-seq experiments. It is based on the insights published by Diaz et al.



Once we're satisfied with the basic quality checks, we normally **convert** the large *BAM* files into a leaner data format, typically *bigWig*. bigWig files have several advantages over BAM files, mainly stemming from their significantly decreased size:

- useful for data sharing and storage
- intuitive visualization in Genome Browsers (e.g. IGV)
- more efficient downstream analyses are possible

The deepTools modules tools/bamCompare and tools/bamCoverage not only allow for simple conversion of BAM to bigWig (or *bedGraph* for that matter), but also for normalization, such that different samples can be compared despite differences in their sequencing depth.

Finally, once all the converted files have passed our visual inspections (e.g., using the Integrative Genomics Viewer), the fun of downstream analysis with tools/computeMatrix, tools/plotHeatmap and tools/plotProfile can begin!

deepTools Galaxy.	code @ github.

1.5 Changes in deepTools2.0

- *Major changes*
  - *Accommodating additional data types*
  - *Structural updates*
  - *Renamed tools*
  - *Increased efficiency*
  - *New features and tools*
- *Minor changes*

  - *Changed parameters names and settings*
  - *Bug fixes*

1.5.1 Major changes

Note: The major changes encompass features for **increased efficiency**, **new sequencing data types**, and **additional plots**, particularly for QC.

Moreover, deepTools modules can now be used by other python programs. The *deepTools API example* is part of the new documentation.

Accommodating additional data types

- correlation and comparisons can now be calculated for **bigWig files** (in addition to BAM files) using `multiBigwigSummary` and `bigwigCompare`
- **RNA-seq:** split-reads are now natively supported
- **MNase-seq:** using the new option `--MNase` in `bamCoverage`, one can now compute read coverage only taking the 2 central base pairs of each mapped fragment into account.

Structural updates

- All modules have comprehensive and automatic tests that evaluate proper functioning after any modification of the code.
- Virtualization for stability: we now provide a `docker` image and enable the easy deployment of deepTools via the Galaxy `toolshed`.
- Our documentation is now version-aware thanks to readthedocs and `sphinx`.
- The API is public and documented.

Renamed tools

- **heatmapper** to tools/plotHeatmap
- **profiler** to tools/plotProfile
- **bamCorrelate** to tools/multiBamSummary
- **bigwigCorrelate** to tools/multiBigwigSummary
- **bamFingerprint** to tools/plotFingerprint

Increased efficiency

- We dramatically improved the **speed** of bigwig related tools (tools/multiBigwigSummary and `computeMatrix`) by using the new pyBigWig module.
- It is now possible to generate one composite heatmap and/or meta-gene image based on **multiple bigwig files** in one go (see tools/computeMatrix, tools/plotHeatmap, and tools/plotProfile for examples)

- `computeMatrix` now also accepts multiple input BED files. Each is treated as a group within a sample and is plotted independently.
- We added **additional filtering options for handling BAM files**, decreasing the need for prior filtering using tools other than deepTools: The `--samFlagInclude` and `--samFlagExclude` parameters can, for example, be used to only include (or exclude) forward reads in an analysis.
- We separated the generation of read count tables from the calculation of pairwise correlations that was previously handled by `bamCorrelate`. Now, read counts are calculated first using `multiBamSummary` or `multiBigWigCoverage` and the resulting output file can be used for calculating and plotting pairwise correlations using `plotCorrelation` or for doing a principal component analysis using `plotPCA`.

New features and tools

- Correlation analyses are no longer limited to BAM files – bigwig files are possible, too! (see tools/multiBigwigSummary)
- Correlation coefficients can now be computed even if the data contains NaNs.
- **Added new quality control tools:**
    - use tools/plotCoverage to plot the coverage over base pairs
    - use tools/plotPCA for principal component analysis
    - tools/bamPEFragmentSize can be used to calculate the average fragment size for paired-end read data
- Added the possibility for **hierarchical clustering**, besides *k*-means to `plotProfile` and `plotHeatmap`
- `plotProfile` has many more options to make compelling summary plots

1.5.2 Minor changes

Changed parameters names and settings

- `computeMatrix` can now read files with DOS newline characters.
- `--missingDataAsZero` was renamed to `--skipNonCoveredRegions` for clarity in `bamCoverage` and `bamCompare`.
- Read extension was made optional and we removed the need to specify a default fragment length for most of the tools: `--fragmentLength` was thus replaced by the new optional parameter `--extendReads`.
- Added option `--skipChromosomes` to `multiBigwigSummary`, which can be used to, for example, skip all ‘random’ chromosomes.
- Added the option for adding titles to QC plots.

Bug fixes

- Resolved an error introduced by `numpy version 1.10` in `computeMatrix`.
- Improved plotting features for `plotProfile` when using as plot type: ‘overlapped_lines’ and ‘heatmap’
- Fixed problem with BED intervals in `multiBigwigSummary` and `multiBamSummary` that returned wrongly labeled raw counts.
- `multiBigwigSummary` now also considers chromosomes as identical when the names between samples differ by ‘chr’ prefix, e.g. chr1 vs. 1.

- Fixed problem with wrongly labeled proper read pairs in a BAM file. We now have additional checks to determine if a read pair is a proper pair: the reads must face each other and are not allowed to be farther apart than 4x the mean fragment length.
- For `bamCoverage` and `bamCompare`, the behavior of `scaleFactor` was updated such that now, if given in combination with the normalization options (`--normalizeTo1x` or `--normalizeUsingRPKM`), the given scaling factor will be multiplied with the factor computed by the respective normalization method.

deepTools Galaxy. | code @ github.

1.6 Using deepTools within Galaxy

Galaxy is a tremendously useful platform developed by the Galaxy Team at Penn State and the Emory University. This platform is meant to offer access to a large variety of bioinformatics tools that can be used without computer programming experiences. That means, that the basic features of Galaxy will apply to every tool, i.e. every tool provided within a Galaxy framework will look very similar and will follow the concepts of Galaxy.

Our publicly available deepTools Galaxy instance can be found here: deeptools.ie-freiburg.mpg.de. This server also contains some additional tools that will enable users to analyse and visualize data from high-throughput sequencing experiments, starting from aligned reads.

Table of content

- *Basic features of Galaxy*
  - *The start site*
  - *Details*
  - *Handling failed files*
  - *Workflows*

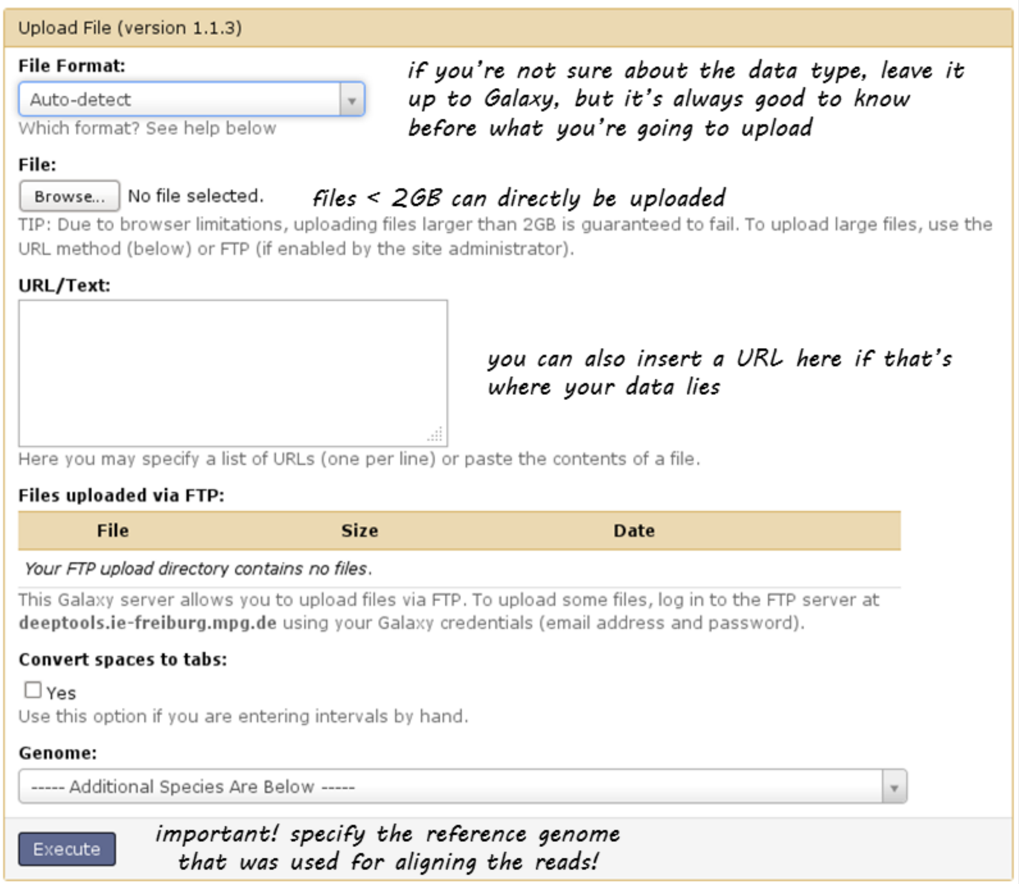
1.6.1 Data import into Galaxy

There are three main ways to populate your Galaxy history with data files plus an additional one for sharing data within Galaxy.

- *Upload files from your computer*
- *Import data sets from the Galaxy data library*
- *Download annotation files from public data bases*
- *Copy data sets between histories*

Upload files from your computer

The data upload of files **smaller than 2 GB** that lie on your computer is fairly straight-forward: click on the category “Get data” and choose the tool “Upload file”. Then select the file via the “Browse” button.



File Format: Auto-detect

Which format? See help below

if you're not sure about the data type, leave it up to Galaxy, but it's always good to know before what you're going to upload

File: No file selected.

files < 2GB can directly be uploaded

TIP: Due to browser limitations, uploading files larger than 2GB is guaranteed to fail. To upload large files, use the URL method (below) or FTP (if enabled by the site administrator).

URL/Text:

you can also insert a URL here if that's where your data lies

Here you may specify a list of URLs (one per line) or paste the contents of a file.

Files uploaded via FTP:

Your FTP upload directory contains no files.

This Galaxy server allows you to upload files via FTP. To upload some files, log in to the FTP server at **deeptools.ie-freiburg.mpg.de** using your Galaxy credentials (email address and password).

Convert spaces to tabs:

☐ Yes

Use this option if you are entering intervals by hand.

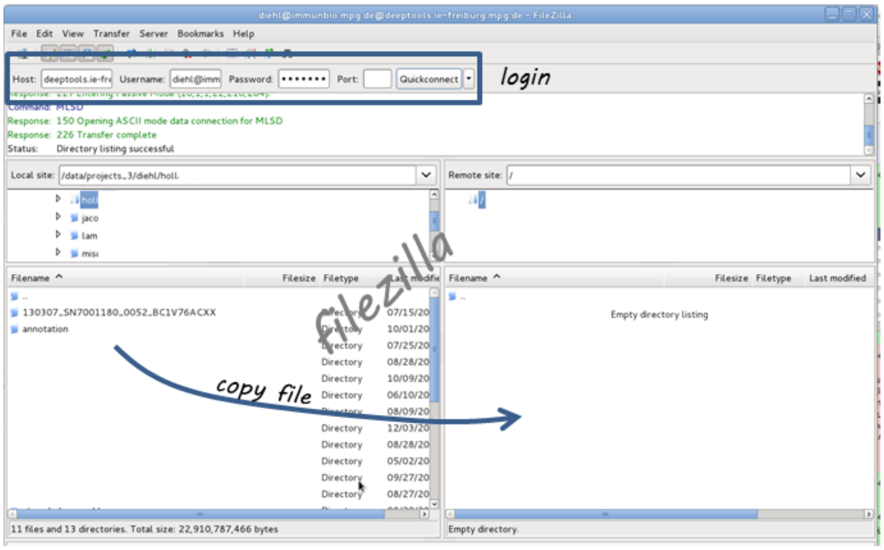
Genome: ----- Additional Species Are Below -----

important! specify the reference genome that was used for aligning the reads!

For files **greater than 2GB**, there's the option to upload via an FTP server. If your data is available via an URL that links to an FTP server, you can simply paste the URL in the empty text box.

If you do not have access to an FTP server, you can directly upload to our Galaxy's FTP.

1. Register with deeptools.ie-freiburg.mpg.de (via “User” “register”; registration requires an email address and is free of charge)
2. You will also need an FTP client, e.g. filezilla.
3. Then login to the **FTP client** using your **deepTools Galaxy user name and password** (host: deeptools.ie-freiburg.mpg.de). Down below you see a screenshot of what that looks like with filezilla.
4. Copy the file you wish to upload to the remote site (in filezilla, you can simply drag the file to the window on the right hand side)
5. Go back to deepTools Galaxy.
6. Click on the tool “Upload file” (“Files uploaded via FTP”) - here, the files you just copied over via filezilla should appear. Select the files you want and hit “execute”. They will be moved from the FTP server to your history (i.e. they will be deleted from the FTP once the upload was successful).


diehl@immunbio.mpg.de@deeptools.ie-freiburg.mpg.de - FileZilla
File Edit View Transfer Server Bookmarks Help
login
Filezilla
copy file
Filename
Filesize
Filetype
Last modified

Host: deeptools.ie-fre Username: diehl@imm Password: ••••••• Port: Quickconnect

```
Command: MLSD
Response: 150 Opening ASCII mode data connection for MLSD
Response: 226 Transfer complete
Status:   Directory listing successful
```

Local site: /data/projects_3/diehl/holl.

Remote site: /

11 files and 13 directories. Total size: 22,910,787,466 bytes

Empty directory listing

Empty directory.

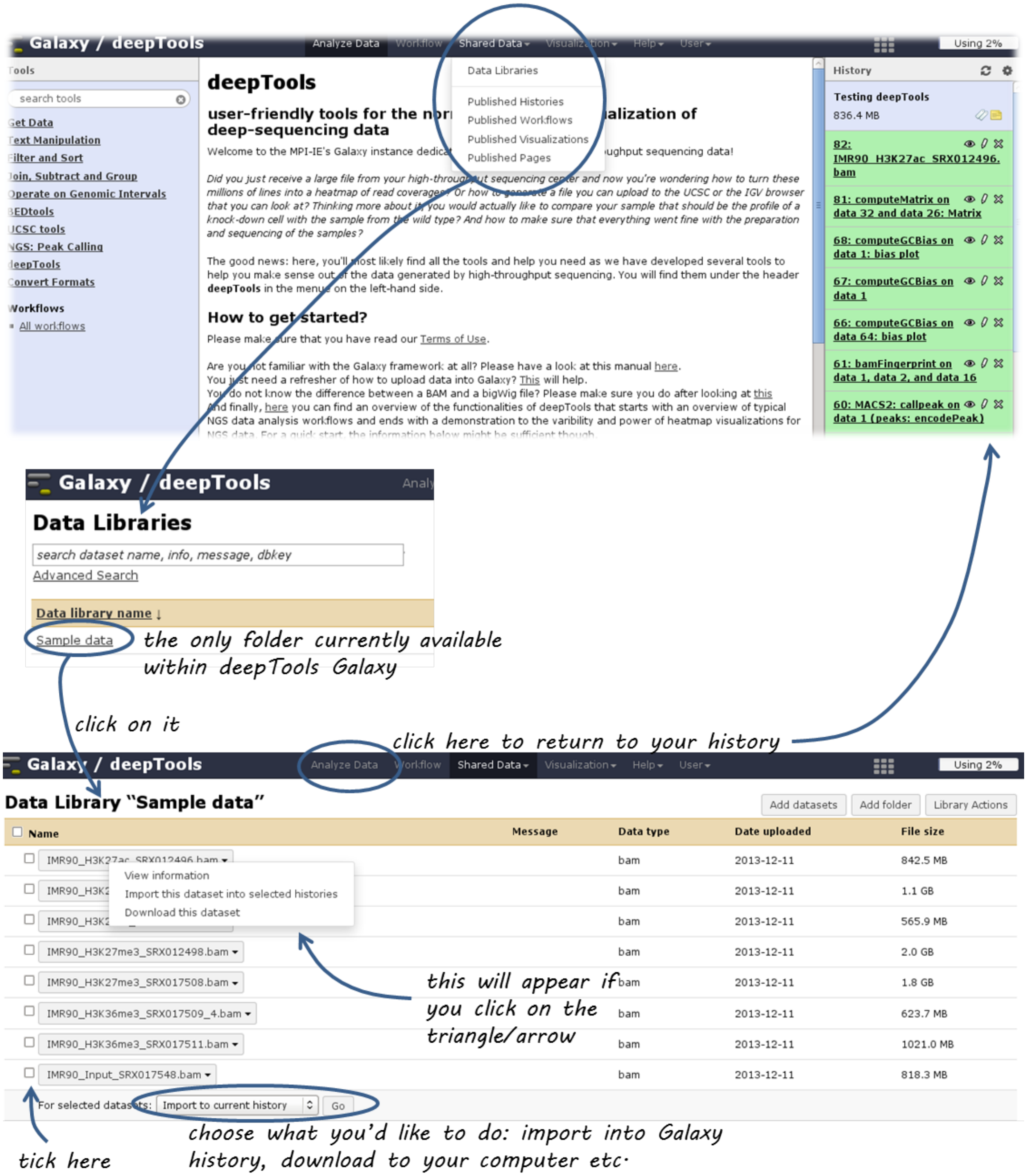
Import data sets from the Galaxy data library

If you would like to play around with sample data, you can import files that we have saved within the general data storage of the deepTools Galaxy server. Everyone can import them into his or her own history, they will not contribute to the user's disk quota.

You can reach the data library via "Shared Data" in the top menu, then select "Data Libraries".

Within the Data Library you will find a folder called "Sample Data" that contains data that we downloaded from the Roadmap project and UCSC More precisely, we donwloaded the [FASTQ][] files of various ChIP-seq samples and the corresponding input and mapped the reads to the human reference genome (version hg19) to obtain the [BAM][] files you see. In addition, you will find bigWig files created using `bamCoverage` and some annotation files in BED format as well as RNA-seq data.

Note: To keep the file size smallish, all files contain data for chromosome 19 and chromosome X only!


Galaxy / deepTools
Analyze Data
Workflow
Shared Data
Visualization
Help
User
Using 2%
Tools
search tools
et Data
ext Manipulation
ilter and Sort
oin, Subtract and Group
perate on Genomic Intervals
EDtools
CSC tools
GS: Peak Calling
eepTools
onvert Formats
orkflows
All workflows
Data Libraries
Published Histories
Published Workflows
Published Visualizations
Published Pages
History
Testing deepTools
836.4 MB
82: IMR90_H3K27ac_SRX012496.bam
81: computeMatrix on data 32 and data 26: Matrix
68: computeGCBias on data 1: bias plot
67: computeGCBias on data 1
66: computeGCBias on data 64: bias plot
61: bamFingerprint on data 1, data 2, and data 16
60: MACS2: callpeak on data 1 (peaks: encodePeak)

deepTools

user-friendly tools for the nor alization of deep-sequencing data

Welcome to the MPI-IE's Galaxy instance dedica ughput sequencing data!

Did you just receive a large file from your high-throughput sequencing center and now you're wondering how to turn these millions of lines into a heatmap of read coverages? Or how to generate a file you can upload to the UCSC or the IGV browser that you can look at? Thinking more about it, you would actually like to compare your sample that should be the profile of a knock-down cell with the sample from the wild type? And how to make sure that everything went fine with the preparation and sequencing of the samples?

The good news: here, you'll most likely find all the tools and help you need as we have developed several tools to help you make sense out of the data generated by high-throughput sequencing. You will find them under the header **deepTools** in the menue on the left-hand side.

How to get started?

Please make sure that you have read our Terms of Use.

Are you not familiar with the Galaxy framework at all? Please have a look at this manual here.
You just need a refresher of how to upload data into Galaxy? This will help.
You do not know the difference between a BAM and a bigWig file? Please make sure you do after looking at this
And finally, here you can find an overview of the functionalities of deepTools that starts with an overview of typical NGS data analysis workflows and ends with a demonstration to the varibility and power of heatmap visualizations for NGS data. For a quick start, the information below might be sufficient though.

Galaxy / deepTools

Data Libraries

search dataset name, info, message, dbkey

Advanced Search

Data library name ↓

Sample data

the only folder currently available within deepTools Galaxy

click on it

click here to return to your history

Data Library "Sample data"

Add datasets
Add folder
Library Actions
View information
Import this dataset into selected histories
Download this dataset

☐ Name	Message	Data type	Date uploaded	File size
☐ IMR90_H3K27ac_SRX012496.bam ▾		bam	2013-12-11	842.5 MB
☐ IMR90_H3K2		bam	2013-12-11	1.1 GB
☐ IMR90_H3K2		bam	2013-12-11	565.9 MB
☐ IMR90_H3K27me3_SRX012498.bam ▾		bam	2013-12-11	2.0 GB
☐ IMR90_H3K27me3_SRX017508.bam ▾		bam	2013-12-11	1.8 GB
☐ IMR90_H3K36me3_SRX017509_4.bam ▾		bam	2013-12-11	623.7 MB
☐ IMR90_H3K36me3_SRX017511.bam ▾		bam	2013-12-11	1021.0 MB
☐ IMR90_Input_SRX017548.bam ▾		bam	2013-12-11	818.3 MB

For selected datasets: Import to current history Go

this will appear if you click on the triangle/arrow

tick here

choose what you'd like to do: import into Galaxy history, download to your computer etc.

Download annotation files from public data bases

In many cases you will want to query your sequencing data results for known genome annotation, such as genes, exons, transcription start sites etc. These information can be obtained via the two main sources of genome annotation, UCSC and BioMart.

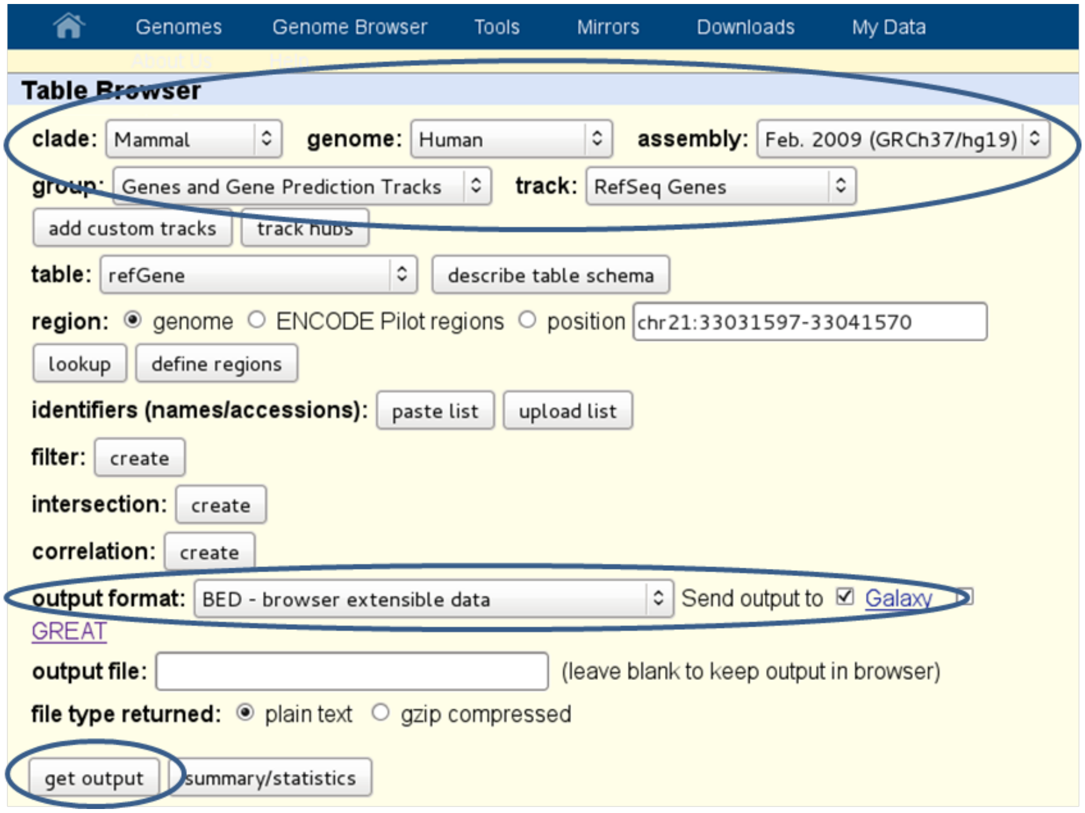
> **Warning:** UCSC and BioMart cater to different ways of genome annotation, i.e. genes defined in UCSC might not correspond to the same regions in a gene file downloaded from BioMart. (For a brief overview over the issues of genome annotation, you can check out Wikipedia, if you always wanted to know much more about those issues, this might be a good start.)

You can access the data stored at UCSC or BioMart conveniently through our Galaxy instance which will import the resulting files into your history. Just go to **“Get data”** “UCSC” or “BioMart”.

The majority of annotation files will probably be in [BED][] format, however, you can also find other data sets. UCSC, for example, offers a wide range of data that you can browse via the “group” and “track” menus (for example, you could download the GC content of the genome as a signal file from UCSC via the “group” menu (“Mapping and Sequencing Tracks”).

> **Warning:** The download through this interface is limited to 100,000 lines per file which might not be sufficient for some mammalian data sets.

Here’s a screenshot from downloading a BED-file of all RefSeq genes defined for the human genome (version hg19):



Genomes
Genome Browser
Tools
Mirrors
Downloads
My Data
add custom tracks
track hubs
describe table schema
lookup
define regions
paste list
upload list
create
create
create
GREAT
get output
summary/statistics

Table Browser

clade: Mammal genome: Human assembly: Feb. 2009 (GRCh37/hg19)

group: Genes and Gene Prediction Tracks track: RefSeq Genes

table: refGene

region: ● genome ○ ENCODE Pilot regions ○ position chr21:33031597-33041570

identifiers (names/accessions):

filter:

intersection:

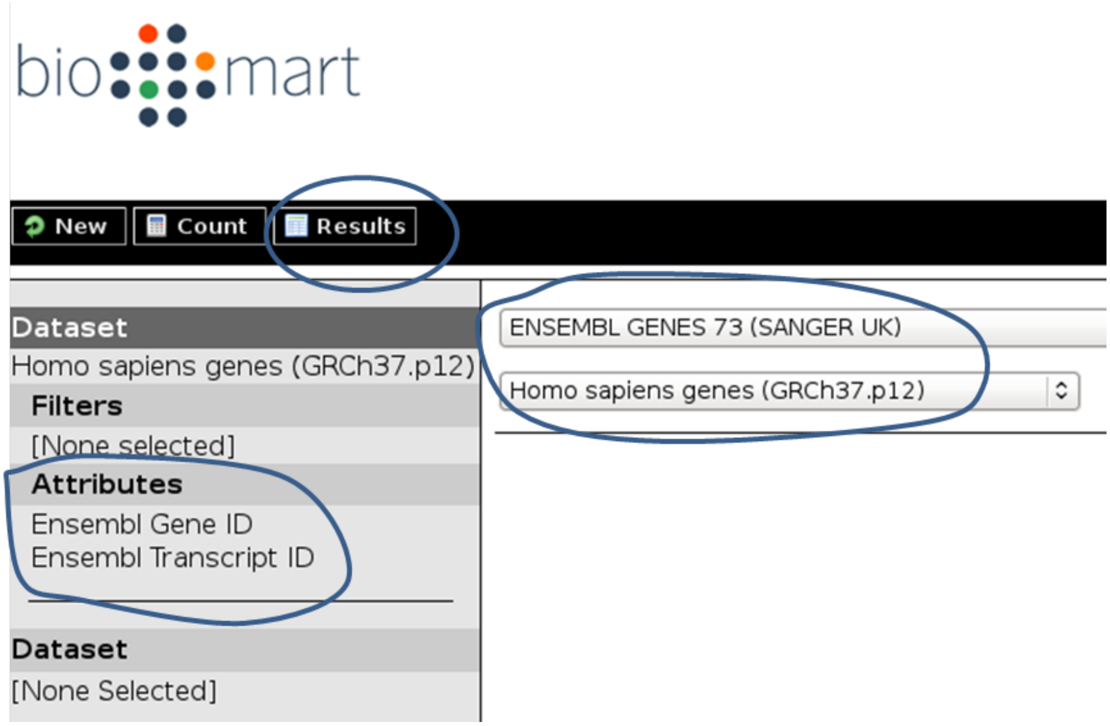
correlation:

output format: BED - browser extensible data Send output to ☑ Galaxy ☐ GREAT

output file: (leave blank to keep output in browser)

file type returned: ● plain text ○ gzip compressed

And here’s how you would do it for the BioMart approach:



Dataset
Homo sapiens genes (GRCh37.p12)
Filters
[None selected]
Attributes
Ensembl Gene ID
Ensembl Transcript ID
Dataset
[None Selected]

ENSEMBL GENES 73 (SANGER UK)
Homo sapiens genes (GRCh37.p12)

Tip: Per default, **BioMart will not output a BED file** like UCSC does. It is therefore important that you make sure you get all the information you need (most likely: chromosome, gene start, gene end, ID, strand) via the "Attributes" section. You can click on the "Results" button at any time to check the format of the table that will be sent to Galaxy (Note that the strand information will be decoded as 1 for "forward" or "plus" strand and -1 for "reverse" or "minus" strand).

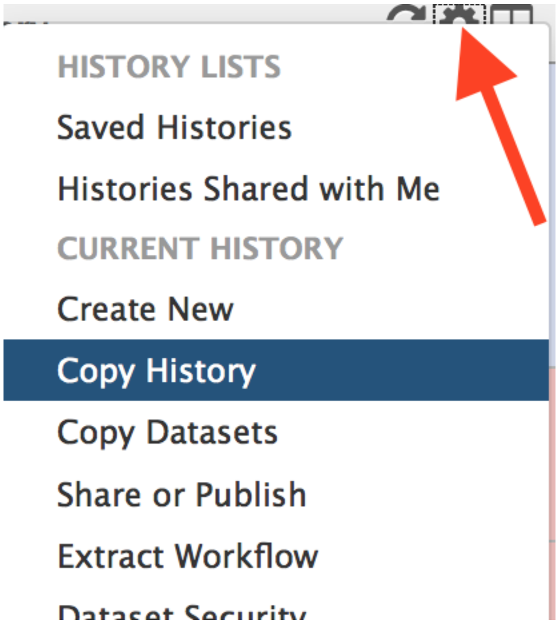
Warning: Be aware, that BED files from UCSC will have chromosomes labelled with "chr" while ENSEMBL usually returns just the number – this might lead to incompatibilities, i.e. when working with annotations from UCSC and ENSEMBL, you need to make sure to use the same naming!

Copy data sets between histories

If you have registered with deepTools Galaxy you can have more than one history.

In order to minimize the disk space you're occupying we strongly suggest to **copy** data sets between histories when you're using the same data set in different histories.

Note: Copying data sets is only possible for registered users.


HISTORY LISTS
Saved Histories
Histories Shared with Me
CURRENT HISTORY
Create New
Copy History
Copy Datasets
Share or Publish
Extract Workflow
Dataset Security

Copying can easily be done via the History panel's `option` button "Copy dataset". In the main frame, you should now be able to select the history you would like to copy from on the left hand side and the target history on the right hand side.

More help

Hint: If you encounter a failing data set (marked in red), please **send a bug report** via the Galaxy bug report button and we will get in touch if you indicate your email address.

http://wiki.galaxyproject.org/Learn	Help for Galaxy usage in general
deepTools Galaxy FAQs	Frequently encountered issues with our specific Galaxy instance
Biostars	For issues not addressed in the FAQs

deepTools Galaxy.	code @ github.

1.6.2 Which tools can I find in the deepTools Galaxy?

As mentioned before, each Galaxy installation can be tuned to the individual interests. Our goal is to provide a Galaxy that enables you to **quality check, process and normalize and subsequently visualize your data obtained by high-throughput DNA sequencing**.

Tip: If you do not know the difference between a BAM and a BED file, that's fine. You can read up on them in our *Glossary of NGS terms*.

Tip: For more specific help, check our *Galaxy-related FAQ* and the *Step-by-step protocols*.

We provide the following kinds of tools:

- *deepTools*
    - *Tools for BAM and bigWig file processing*
    - *Tools for QC of NGS data*
    - *Heatmaps and summary plots*
- *Working with text files and tables*
    - *Text manipulation*
    - *Filter and Sort*
    - *Join, Subtract, Group*
- *Basic arithmetics for tables*

deepTools

The most important category is called **"deepTools"** that contains all the main tools we have developed.

Tools for BAM and bigWig file processing

`multiBamSummary`	get read counts for the binned genome or user-specified regions
`multiBigwigSummary`	calculate score summaries for the binned genome or user-specified regions
`correctGCBias`	obtain a BAM file with reads distributed according to the genome's GC content
`bamCoverage`	obtain the normalized read coverage of a single BAM file
`bamCompare`	normalize 2 BAM files to each other (e.g. log2ratio, difference)
`bigwigCompare`	normalize the scores of two bigWig files to each other (e.g., ratios)
`computeMatrix`	compute the values needed for heatmaps and summary plots

Tools for QC of NGS data

`plotCorrelation`	calculate and visualize the pairwise Spearman or Pearson correlation of read counts (or other scores)
`plotPCA`	perform PCA and visualize the results
`plotFingerprint`	assess the ChIP enrichment strength
`bamPEFragmentSize`	obtain the average fragment length for paired-end samples
`computeGCBias`	assess the GC bias by calculating the expected and observed GC distribution of aligned reads
`plotCoverage`	obtain the normalized read coverage of a single BAM file

Heatmaps and summary plots

`plotHeatmap`	visualize read counts or other scores in heatmaps with one row per genomic region
`plotProfile`	visualize read counts or other scores using average profiles (e.g., meta-gene profiles)

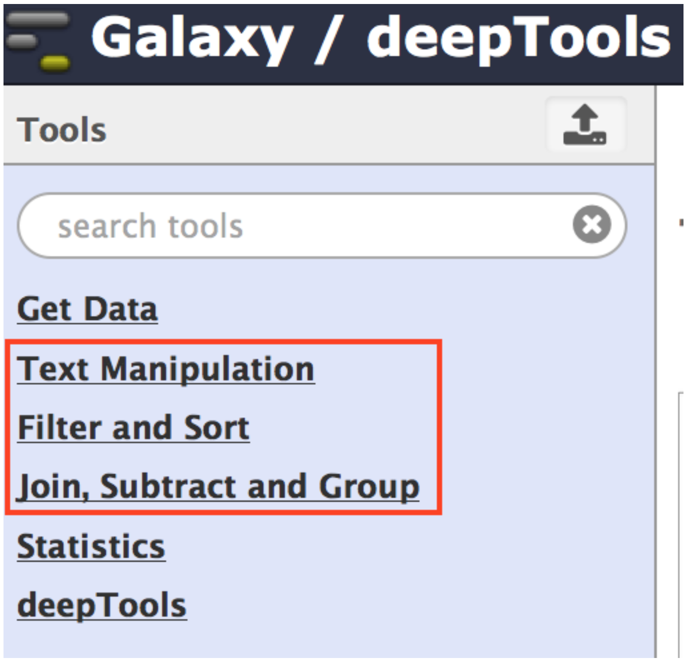
For each tool, you can find example usages and tips within Galaxy once you select the tool.

In addition, you may want to check our pages about *Example usage*, particularly *Step-by-step protocols*.

Working with text files and tables

In addition to deepTools that were specifically developed for the handling of NGS data, we have incorporated several standard Galaxy tools that enable you to manipulate tab-separated files such as gene lists, peak lists, data matrices etc.

There are 3 main categories;



Text manipulation

Unlike Excel, where you can easily interact with your text and tables via the mouse, data manipulations within Galaxy are strictly based on commands.

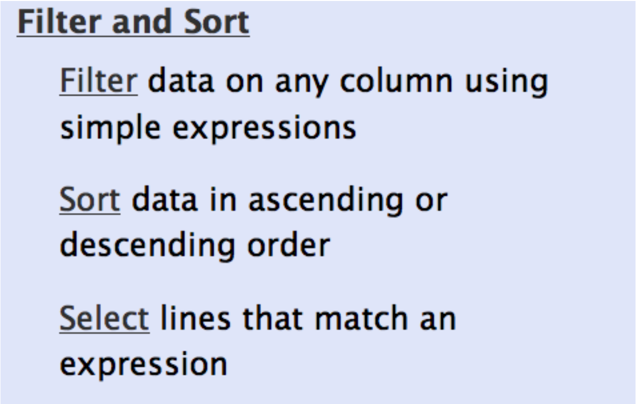
If you feel like you would like to do something to certain *columns* of a data set, go through the tools of this category!

Example actions are: * adding columns * extracting columns * pasting two files side by side * selecting random lines * etc.

A very useful tool of this category is called `Trim`: if you need to **remove some characters from a column**, this tool's for you! (for example, sometimes you need to adjust the chromosome naming between two files from different source - using `Trim`, you can remove the "chr" in front of the chromosome name)

Filter and Sort

In addition to the common sorting and filtering, there's the very useful tool to `select lines that match an expression`. For example, using the expression `c1=='chrM'` will select all rows from a BED file with regions located on the mitochondrial chromosome.



Filter and Sort

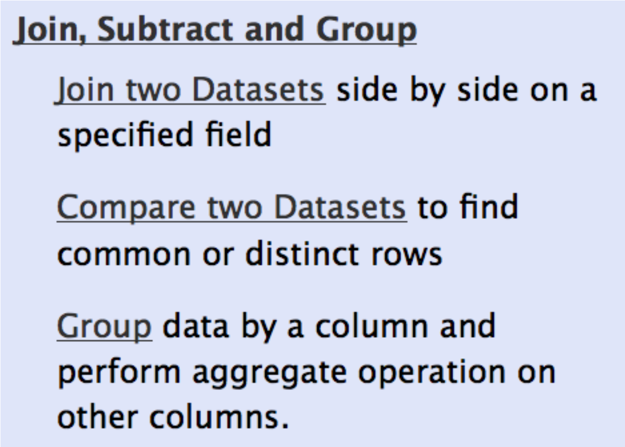
Filter data on any column using simple expressions

Sort data in ascending or descending order

Select lines that match an expression

Join, Subtract, Group

The tools of this category are very useful if you have several data sets that you would like to work with, e.g. by comparing them.



Join, Subtract and Group

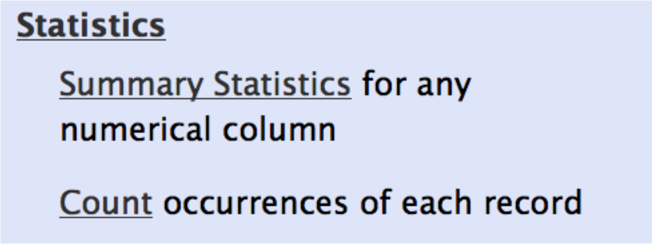
Join two Datasets side by side on a specified field

Compare two Datasets to find common or distinct rows

Group data by a column and perform aggregate operation on other columns.

Basic arithmetics for tables

We offer some very basic mathematical operations on values stored with tables. The `Summary Statistics` can be used to calculate the sum, mean, standard deviation and percentiles for a set of numbers, e.g. for values stored in a specific column.



Statistics

Summary Statistics for any numerical column

Count occurrences of each record

More help

Hint: If you encounter a failing data set (marked in red), please **send a bug report** via the Galaxy bug report button and we will get in touch if you indicate your email address.

---

http://wiki.galaxyproject.org/Learn	Help for Galaxy usage in general
deepTools Galaxy FAQs	Frequently encountered issues with our specific Galaxy instance
Biostars	For issues not addressed in the FAQs

deepTools Galaxy.	code @ github.

1.6.3 Basic features of Galaxy

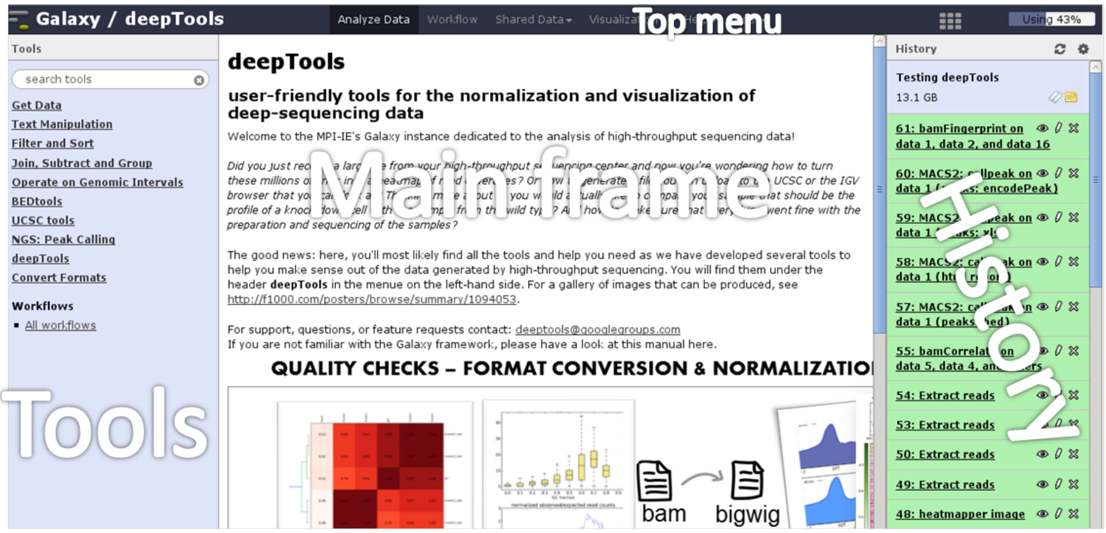
Galaxy is a web-based platform for data intensive, bioinformatics-dependent research and it is being developed by Penn State and John Hopkins University. The original Galaxy can be found here.

Since it is impossible to meet all bioinformatics needs – that can range from evolutionary analysis to data from mass spectrometry to high-throughput DNA sequencing (and way beyond) – with one single web server, many institutes have installed their own versions of the Galaxy platform tuned to their specific needs.

Our deepTools Galaxy is such a specialized server dedicated to the analysis of high-throughput DNA sequencing data. The overall makeup of this web server, however, is the same as for any other Galaxy installation, so if you've used Galaxy before, you will learn to use deepTools in no time!

The start site

Here is a screenshot of what the start site will approximately look like:


Galaxy / deepTools
Analyze Data
Workflow
Shared Data
Visualiza
Top menu
Using 43%
Tools
search tools
Get Data
Text Manipulation
Filter and Sort
Join, Subtract and Group
Operate on Genomic Intervals
BEDtools
UCSC tools
NGS: Peak Calling
deepTools
Convert Formats
Workflows
All workflows
Tools
Main frame
History
History
Testing deepTools
13.1 GB
61: bamFingerprint on data 1, data 2, and data 16
60: MACS2: callpeak on data 1
59: MACS2
58: MACS2
57: MACS2
55: bamCorrelate on data 5, data 4, an
54: Extract reads
53: Extract reads
50: Extract reads
49: Extract reads
48: heatmapper image
QUALITY CHECKS – FORMAT CONVERSION & NORMALIZATIO
bam
bigwig

deepTools

user-friendly tools for the normalization and visualization of deep-sequencing data

Welcome to the MPI-IE's Galaxy instance dedicated to the analysis of high-throughput sequencing data!

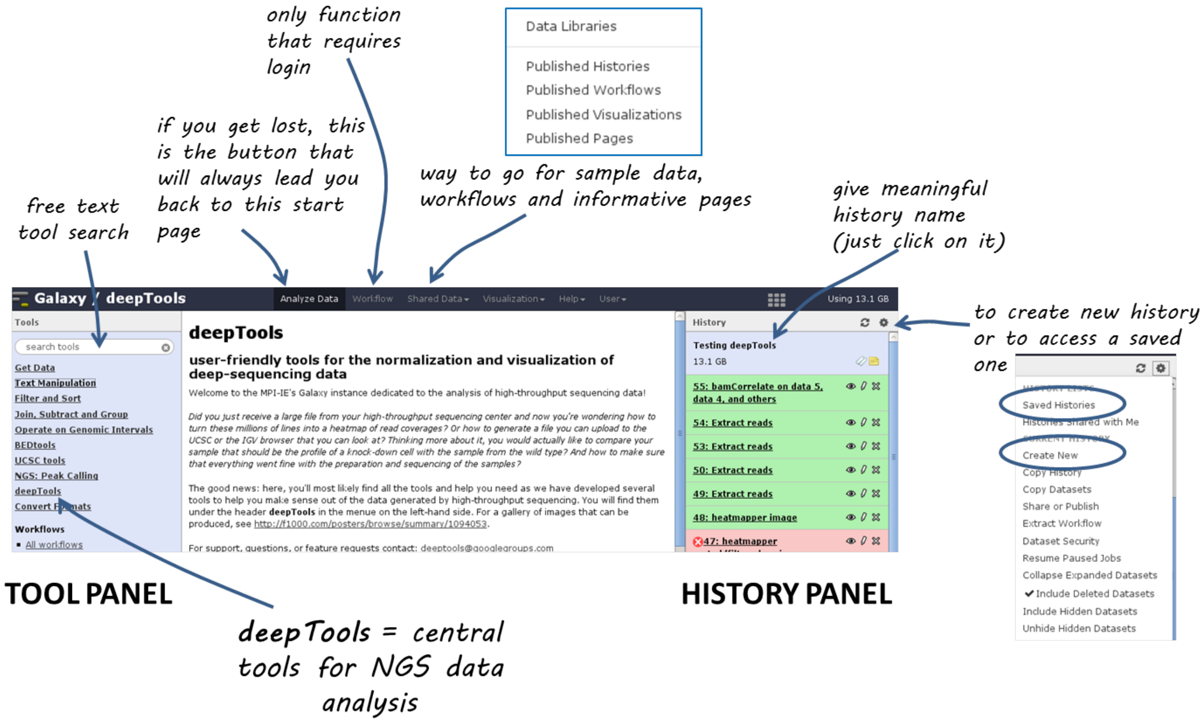
The good news: here, you'll most likely find all the tools and help you need as we have developed several tools to help you make sense out of the data generated by high-throughput sequencing. You will find them under the header **deepTools** in the menue on the left-hand side. For a gallery of images that can be produced, see http://f1000.com/posters/browse/summary/1094053.

For support, questions, or feature requests contact: deeptools@googlegroups.com
If you are not familiar with the Galaxy framework, please have a look at this manual here.

The start site contains 4 main features:

Top menu	Your gateway away from the actual analysis part into other sections of Galaxy, e.g. **workflows** and **shared data**.
Tool panel	*What can be done?* Find all tools installed in this Galaxy instance
Main frame	*What am I doing?* This is your main **working space** where input will be required from you once you've selected a tool.
History panel	*What did I do?* The history is like a **log book**: everything you ever did is recorded here.

For those visual learners, here's an annotated screenshot:



deepTools

user-friendly tools for the normalization and visualization of deep-sequencing data

Welcome to the MPI-IE's Galaxy instance dedicated to the analysis of high-throughput sequencing data!

Did you just receive a large file from your high-throughput sequencing center and now you're wondering how to turn these millions of lines into a heatmap of read coverages? Or how to generate a file you can upload to the UCSC or the IGV browser that you can look at? Thinking more about it, you would actually like to compare your sample that should be the profile of a knock-down cell with the sample from the wild type? And how to make sure that everything went fine with the preparation and sequencing of the samples?

The good news: here, you'll most likely find all the tools and help you need as we have developed several tools to help you make sense out of the data generated by high-throughput sequencing. You will find them under the header **deepTools** in the menue on the left-hand side. For a gallery of images that can be produced, see http://f1000.com/posters/browse/summary/1094053.

For support, questions, or feature requests contact: deeptools@googlegroups.com

only function
that requires
login
Data Libraries
Published Histories
Published Workflows
Published Visualizations
Published Pages
if you get lost, this
is the button that
will always lead you
back to this start
page
way to go for sample data,
workflows and informative pages
give meaningful
history name
(just click on it)
free text
tool search
Galaxy / deepTools
Analyze Data
Workflow
Shared Data
Visualization
Help
User
Using 13.1 GB
Tools
search tools
Get Data
Text Manipulation
Filter and Sort
Join, Subtract and Group
Operate on Genomic Intervals
BEDtools
UCSC tools
NGS: Peak Calling
deepTools
Convert Formats
Workflows
All workflows
History
Testing deepTools
13.1 GB
55: bamCorrelate on data 5, data 4, and others
54: Extract reads
53: Extract reads
50: Extract reads
49: Extract reads
48: heatmapper image
47: heatmapper
to create new history
or to access a saved
one
Saved Histories
Histories Shared with Me
Create New
Copy History
Copy Datasets
Share or Publish
Extract Workflow
Dataset Security
Resume Paused Jobs
Collapse Expanded Datasets
✓ Include Deleted Datasets
Include Hidden Datasets
Unhide Hidden Datasets
TOOL PANEL
HISTORY PANEL
deepTools = central
tools for NGS data
analysis

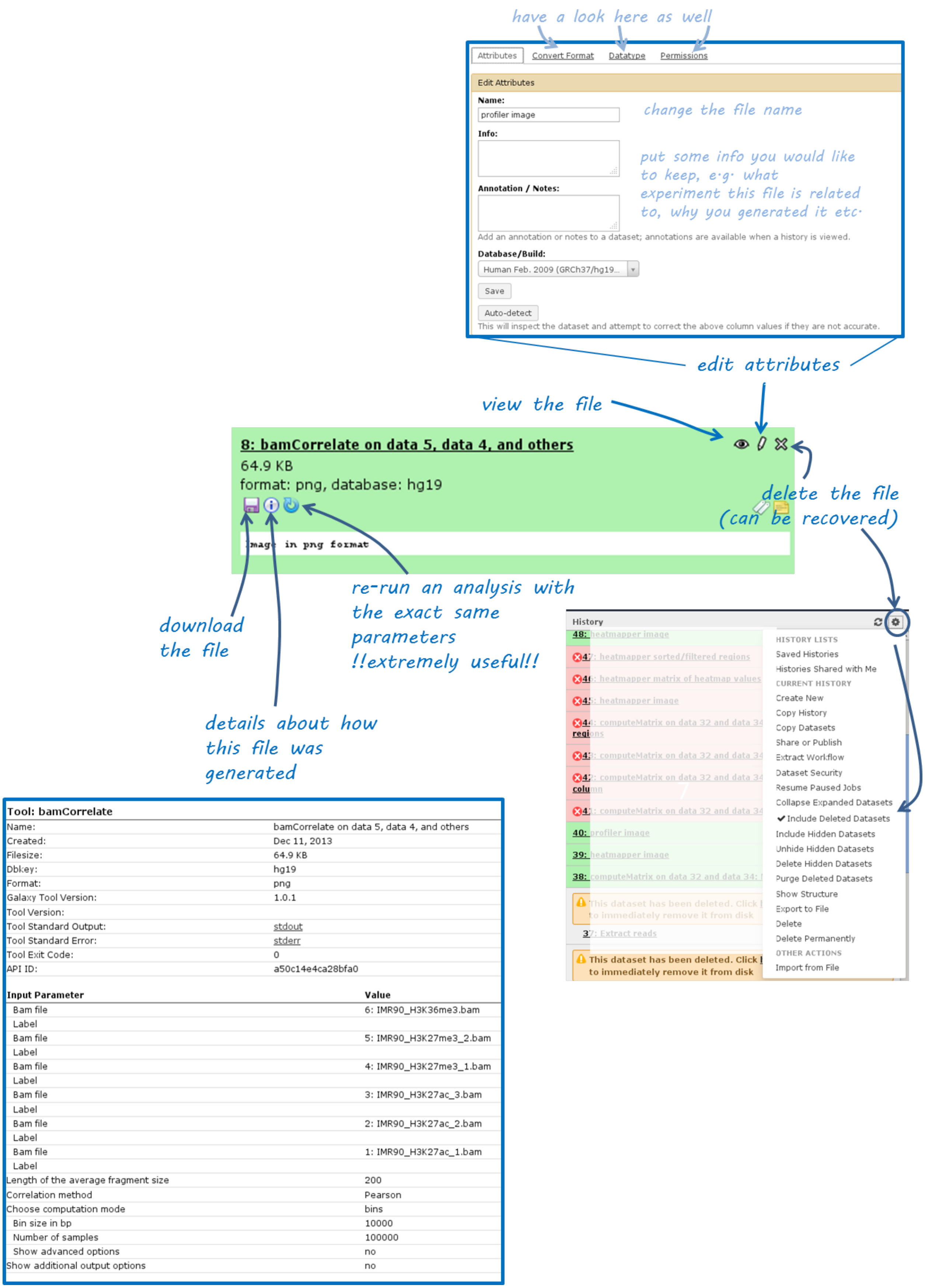
Details

In the default state of the tool panel you see the **tool categories**, e.g. "Get Data". If you click on them, you will see the **individual tools** belonging to each category, e.g. "Upload File from your computer", "UCSC Main table browser" and "Biomart central server" in case you clicked on "Get Data". To use a tool such as "Upload File from your computer", just click on it.

The **tool *search* panel** is extremely useful as it allows you to enter a key word (e.g. "bam") that will lead to all the tools mentioning the key word in the tool name.

Once you've uploaded any kind of data, you will find the history on the right hand side filling up with green tiles. Each tile corresponds to one data set that you either uploaded or created. The data sets can be images, raw sequencing files, text files, tables - virtually anything. The content of a data set *cannot* be modified - every time you want to change something *within* a data file (e.g. you would like to sort the values or add a line or cut a column), you will have to use a Galaxy tool that will lead to a *new* data set being produced. This behaviour is often confusing for Galaxy novices (as histories tend to accumulate data sets very quickly), but it is necessary to enforce the strict policy of documenting *every modification* to a given data set. Eventhough your history might be full of data sets with strange names, you will always be able to track back the source and evolution of each file. Also, every data set can be downloaded to your computer individually. Alternatively, you can *download* an entire history or *share* the history with another user.

Have a look at the following screenshot to get a feeling for how many information Galaxy keeps for you (which makes it very feasible to reproduce any analysis):


Attributes
Convert Format
Datatype
Permissions
Edit Attributes
Save
Auto-detect
History
HISTORY LISTS
Saved Histories
Histories Shared with Me
CURRENT HISTORY
Create New
Copy History
Copy Datasets
Share or Publish
Extract Workflow
Dataset Security
Resume Paused Jobs
Collapse Expanded Datasets
✓ Include Deleted Datasets
Include Hidden Datasets
Unhide Hidden Datasets
Delete Hidden Datasets
Purge Deleted Datasets
Show Structure
Export to File
Delete
Delete Permanently
OTHER ACTIONS
Import from File
48: heatmapper image
47: heatmapper sorted/filtered regions
46: heatmapper matrix of heatmap values
45: heatmapper image
44: computeMatrix on data 32 and data 34
regions
43: computeMatrix on data 32 and data 34
42: computeMatrix on data 32 and data 34
column
41: computeMatrix on data 32 and data 34
40: profiler image
39: heatmapper image
38: computeMatrix on data 32 and data 34:
37: Extract reads

have a look here as well

Name: profiler image

Info:

Annotation / Notes:

Add an annotation or notes to a dataset; annotations are available when a history is viewed.

Database/Build: Human Feb. 2009 (GRCh37/hg19...

This will inspect the dataset and attempt to correct the above column values if they are not accurate.

change the file name

put some info you would like to keep, e.g. what experiment this file is related to, why you generated it etc.

edit attributes

view the file

8: bamCorrelate on data 5, data 4, and others

64.9 KB

format: png, database: hg19

`Image in png format`

delete the file (can be recovered)

download the file

details about how this file was generated

re-run an analysis with the exact same parameters !!extremely usefull!!

This dataset has been deleted. Click here to immediately remove it from disk

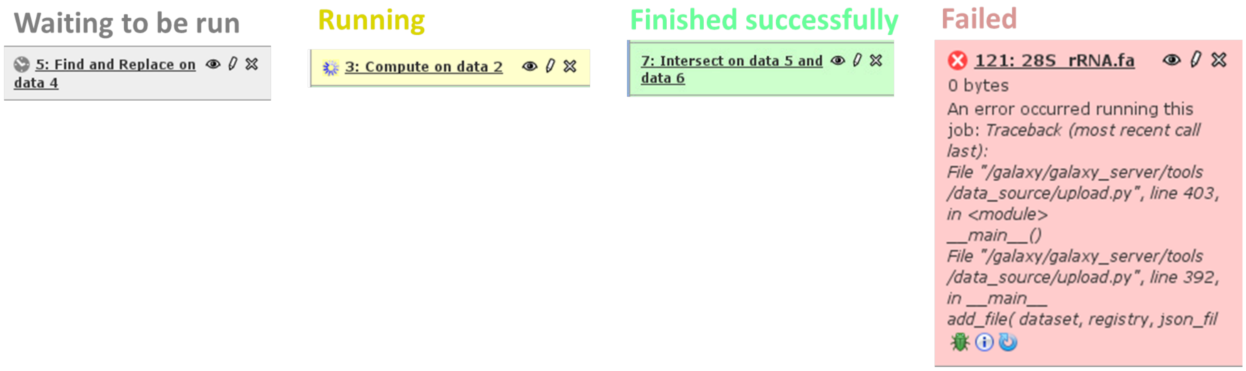
This dataset has been deleted. Click here to immediately remove it from disk

Tool: bamCorrelate

Name:	bamCorrelate on data 5, data 4, and others
Created:	Dec 11, 2013
Filesize:	64.9 KB
Dbkey:	hg19
Format:	png
Galaxy Tool Version:	1.0.1
Tool Version:	
Tool Standard Output:	stdout
Tool Standard Error:	stderr
Tool Exit Code:	0
API ID:	a50c14e4ca28bfa0

Input Parameter	Value
Bam file	6: IMR90_H3K36me3.bam
Label	
Bam file	5: IMR90_H3K27me3_2.bam
Label	
Bam file	4: IMR90_H3K27me3_1.bam
Label	
Bam file	3: IMR90_H3K27ac_3.bam
Label	
Bam file	2: IMR90_H3K27ac_2.bam
Label	
Bam file	1: IMR90_H3K27ac_1.bam
Label	
Length of the average fragment size	200
Correlation method	Pearson
Choose computation mode	bins
Bin size in bp	10000
Number of samples	100000
Show advanced options	no
Show additional output options	no

Each data set can have 4 different states that are intuitively color-coded:


Waiting to be run
5: Find and Replace on data 4
Running
3: Compute on data 2
Finished successfully
7: Intersect on data 5 and data 6
Failed
121: 28S_rRNA.fa

0 bytes

```
An error occurred running this
job: Traceback (most recent call
last):
File "/galaxy/galaxy_server/tools
/data_source/upload.py", line 403,
in <module>
__main__()
File "/galaxy/galaxy_server/tools
/data_source/upload.py", line 392,
in __main__
add_file( dataset, registry, json_fil
```

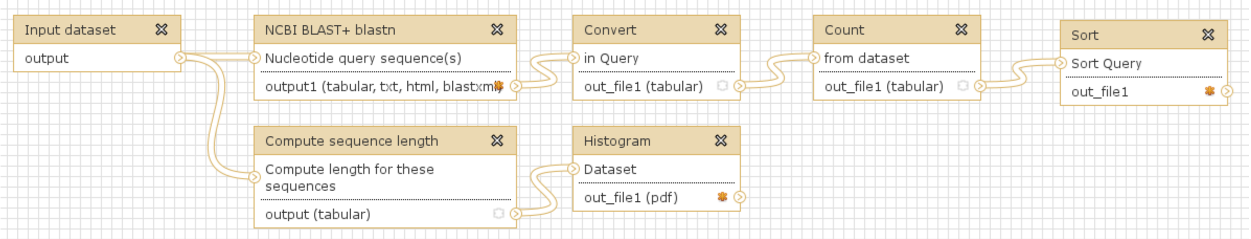
Handling failed files

If you encounter a failed file after you've run a tool, please do the following steps (**in this order**):

1. click on the center button on the lower left corner of the failed data set `(i)`: did you chose the **correct data files**?
2. if you're sure that you chose the correct files, hit the `re-run button` (blue arrow in the lower left corner) - check again whether your files had the **correct file format**. If you suspect that the format might be incorrectly assigned (e.g. a file that should be a BED file is labelled as a tabular file), click the `edit button` (the pencil) of the input data file - there you can change the corresponding attributes
3. if you've checked your input data and the error is persisting, click on the `green bug` (lower left corner of the failed data set) and send the **bug report** to us. You do not need to indicate a valid email-address unless you would like us to get in touch with you once the issue is solved.

Workflows

Workflows are Galaxy's equivalent of protocols. This is a very useful feature as it allows users to *share their protocols and bioinformatic analyses* in a very easy and transparent way. This is the graphical representation of a Galaxy workflow that can easily be modified via drag'n'drop within the workflows manual (you must be registered with deepTools Galaxy to be able to generate your own workflows or edit published ones).


Input dataset
output
NCBI BLAST+ blastn
Nucleotide query sequence(s)
output1 (tabular, txt, html, blastxml
Convert
in Query
out_file1 (tabular)
Count
from dataset
out_file1 (tabular)
Sort
Sort Query
out_file1
Compute sequence length
Compute length for these sequences
output (tabular)
Histogram
Dataset
out_file1 (pdf)

More help

Hint: If you encounter a failing data set (marked in red), please **send a bug report** via the Galaxy bug report button and we will get in touch if you indicate your email address.

---

http://wiki.galaxyproject.org/Learn	Help for Galaxy usage in general
deepTools Galaxy FAQs	Frequently encountered issues with our specific Galaxy instance
Biostars	For issues not addressed in the FAQs

deepTools Galaxy.	code @ github.

1.7 General FAQ

Tip: For support or questions please post to Biostars. For bug reports and feature requests please open an issue <on github.

Note: We also have a *Galaxy-related FAQ* with questions that are more specific to Galaxy rather than deepTools usage.

- *How does deepTools handle data from paired-end sequencing?*
- *How can I test a tool with little computation time?*
- *Can I specify more than one chromosome in the `--regions` option?*
    - *General workaround*
    - *Build-in solutions*
- *When should I exclude regions from `computeGCBias`?*
- *When should I use `bamCoverage` or `bamCompare`?*
- *What should I pay attention to when dealing with RNA-seq data?*
- *How does computeMatrix handle overlapping genome regions?*
    - *Galaxy-based work around*
    - *Command line-based work arounds*
- *Why does the maximum value in the heatmap not equal the maximum value in the matrix?*
- *The heatmap I generated looks very "coarse", I would like a much more fine-grained image.*
- *How can I change the automatic labels of the clusters in a k-means clustered heatmap?*
- *How can I manually specify several groups of regions (instead of clustering)?*
- *What do I have to pay attention to when working with a draft version of a genome?*
- *How do I calculate the effective genome size for an organism that's not in your list?*
    - *Use `GEM`*
    - *Use `faCount`*
    - *Use `bamCoverage`*
    - *Use `genomeCoverageBed`*

- *Where can I download the 2bit genome files required for `computeGCBias`?*

1.7.1 How does deepTools handle data from paired-end sequencing?

Generally, all the modules working on *BAM* files (`multiBamSummary`, `bamCoverage`, `bamCompare`, `plotFingerprint`, `computeGCBias`) automatically recognize paired-end sequencing data and will use the fragment size based on the distance between read pairs. You can by-pass the typical fragment handling on mate pairs with the option `--doNotExtendPairedEnds` (can be found under "advanced options" in Galaxy).

1.7.2 How can I test a tool with little computation time?

When you're playing around with the tools to see what kinds of results they will produce, you can limit the operation to one chromosome or a specific region to save time. In Galaxy, you will find this under "advanced output options" –> "Region of the genome to limit the operation to". The command line option is called `--region` `(CHR:START:END)`.

The following tools currently have this option:

- tools/multiBamSummary
- tools/plotFingerprint
- tools/computeGCBias, tools/correctGCBias
- tools/bamCoverage, tools/bamCompare

It works as follows: first, the *entire* genome represented in the *BAM* file will be regarded and sampled, *then* all the regions or sampled bins that do not overlap the region indicated by the user will be discarded.

Note: You can limit the operation to only **one** chromosome (or **one** specific locus on a chromosome) at a time. If you would like to limit the operation to more than one region, see the answer to the next question.

1.7.3 Can I specify more than one chromosome in the `--regions` option?

The short answer is: no.

Several programs allow specifying a specific regions. For these, the input must be in the format of `chr:start:end`, for example "chr10" or "chr10:456700:891000".

For these programs, it is **not possible** to indicate more than one region, e.g. chr10, chr11 - **this will not work**! Here are some ideas for workarounds if you none-the-less need to do this:

General workaround

Since all the tools that have the `--region` option work on *BAM* files, you could *filter your reads* prior to running the program, e.g. using `intersectBed` with `--abam` or `samtools view`. Then use the resulting (smaller) BAM file with the deepTools program of your choice.

```
$ samtools view -b -L regionsOfInterest.bed Reads.bam >␣
↪ReadsOverlappingWithRegionsOfInterest.bam
```

or

```
$ intersectBed -abam Reads.bam -b regionsOfInterest.bed >␣
↪ReadsOverlappingWithRegionsOfInterest.bam
```

Build-in solutions

`computeGCBias` and `multiBamSummary` offer build-in solutions so that you do not need to resort to tools outside of deepTools.

tools/multiBamSummary has two modes, `bins` and `BED`. If you make use of the `BED` mode, you can supply a *BED* file of regions that you would like to limit the operation to. This will do the same thing as in the general workaround mentioned above.

tools/computeGCBias has a `--filterOut` option. If you to create a BED file that contains all the regions you are **not** interested in, you can then supply this file to `computeGCBias --filterOut Regions_to_be_ignored.bed` and those regions will subsequently be ignored.

1.7.4 When should I exclude regions from `computeGCBias`?

Note: In general, we recommend to only correct for GC bias (using tools/computeGCBias followed by tools/correctGCBias) if the majority of the genome (e.g., for mouse and human genomes the region between 30-60%) is GC-biased *and* you want to compare this sample with another sample that is not GC-biased.

Sometimes, a certain GC bias is expected, for example for ChIP samples of H3K4Me3 in mammalian samples where GC-rich promoters are expected to be enriched. To not confound the GC bias caused by the library preparation with the inherent, expected GC-bias, we incorporated the possibility to supply a file of regions to `computeGCBias` that will be excluded from the GC bias calculation. This file should typically contain those regions that one expects to be significantly enriched. This allows the tool to focus on background regions.

1.7.5 When should I use `bamCoverage` or `bamCompare`?

Both tools produce *bigWig* files, i.e. they translate the read-centered information from a *BAM* file into scores for genomic regions of a fixed size. The only difference is the *number of BAM files* that the tools use as input: while tools/bamCoverage will only take one BAM file and produce a coverage file that is mostly normalized for sequencing depth, tools/bamCompare will take *two BAM* files that can be compared with each other using several mathematical operations.

`bamCompare` will always normalize for sequencing depth like `bamCoverage`, but then it will perform additional calculations depending on what the user chose, for example:

- **ChIP vs.** ***input*** obtain a *bigWig* file of log2ratios(ChIP/input)
- **treatment vs. control** obtain a *bigWig* file of *differences* (treatment - control)
- **replicate 1 and replicate 2** obtain a *bigWig* file where the values from two *BAM* files are summed up (replicate 1 + replicate 2)

1.7.6 What should I pay attention to when dealing with RNA-seq data?

By default, deepTools (**since version 2**) makes use of the information stored in the so-called CIGAR string of the alignment file (SAM/BAM specification). The CIGAR tells precisely to which bases of the reference a read maps - and, accordingly, which bases are skipped in the case of reads that span introns. These so-called split reads are natively handled by all modules of deepTools 2.0.

Warning: It is generally **not** recommended to activate the deepTools parameter `--extendReads` for RNA-seq data.

This is because there is no verified information on the fragment alignment outside the actual read sequence. A simple extension of a read over uncovered parts would probably be wrong for a lot of fragments! Activating the read extension also **deactivates** the utilization of the CIGAR.

1.7.7 How does computeMatrix handle overlapping genome regions?

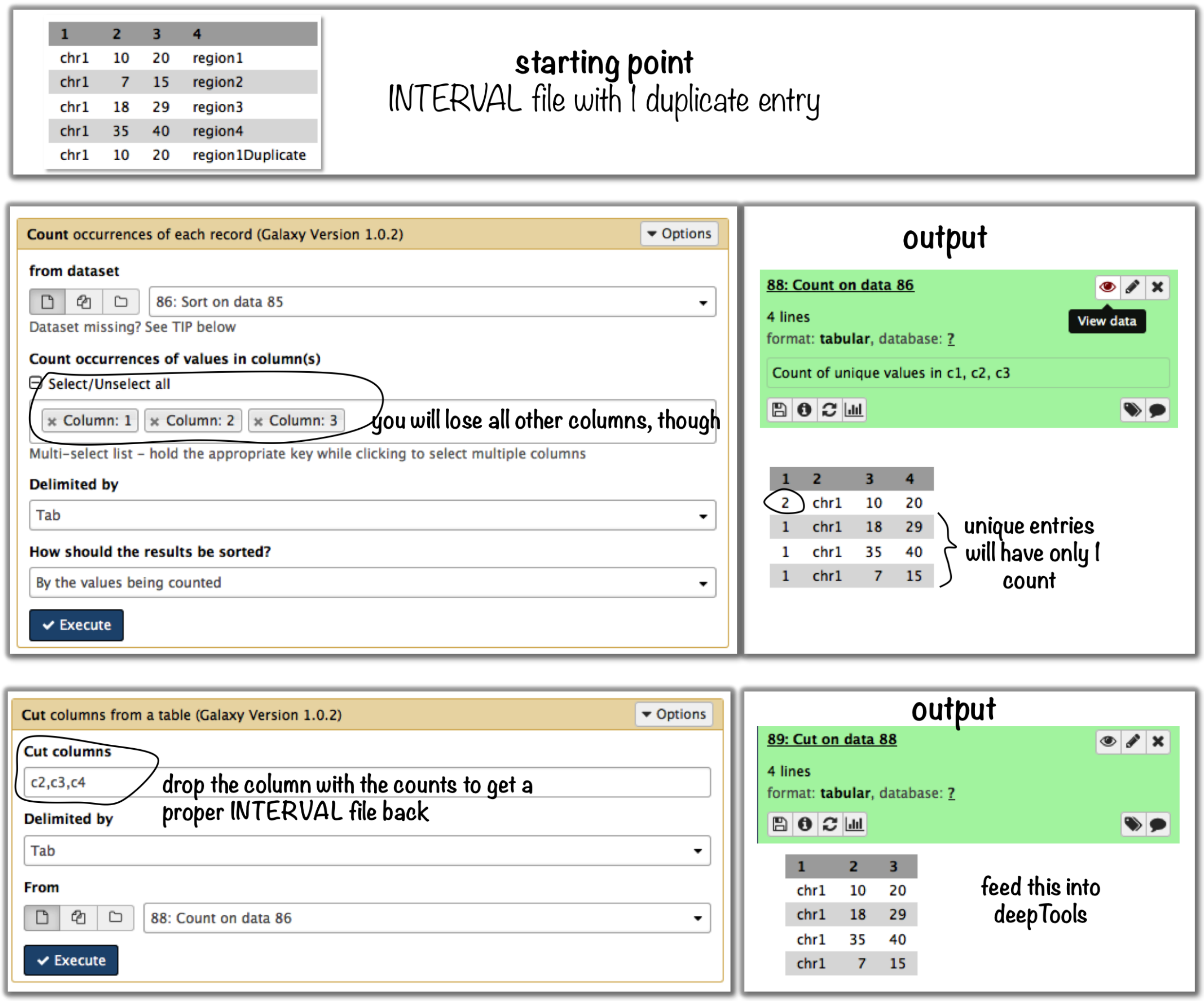
If the *BED* file supplied to tools/computeMatrix contains regions that overlap but they will just be taken as is. If you would like to prevent this, then clean the *BED* file before using `computeMatrix`. There are several methods for modifying your BED file.

Let’s say your file looks like this:

```
$ cat testBed.bed
chr1        10      20      region1
chr1        7       15      region2
chr1        18      29      region3
chr1        35      40      region4
chr1        10      20      region1Duplicate
```

Galaxy-based work around

To eliminate entries with *identical* genome coordinates, first use the tool “Count” and then filter out all entries that are present more than once.



1	2	3	4
chr1	10	20	region1
chr1	7	15	region2
chr1	18	29	region3
chr1	35	40	region4
chr1	10	20	region1Duplicate

starting point
INTERVAL file with 1 duplicate entry

Count occurrences of each record (Galaxy Version 1.0.2)

from dataset: 86: Sort on data 85

Dataset missing? See TIP below

Count occurrences of values in column(s)

Select/Unselect all

Column: 1 Column: 2 Column: 3

you will lose all other columns, though

Multi-select list - hold the appropriate key while clicking to select multiple columns

Delimited by: Tab

How should the results be sorted?: By the values being counted

output

88: Count on data 86

4 lines

format: **tabular**, database: ?

Count of unique values in c1, c2, c3

1	2	3	4
2	chr1	10	20
1	chr1	18	29
1	chr1	35	40
1	chr1	7	15

unique entries will have only 1 count

Cut columns from a table (Galaxy Version 1.0.2)

Cut columns: c2,c3,c4

drop the column with the counts to get a proper INTERVAL file back

Delimited by: Tab

From: 88: Count on data 86

output

89: Cut on data 88

4 lines

format: **tabular**, database: ?

1	2	3
chr1	10	20
chr1	18	29
chr1	35	40
chr1	7	15

feed this into deepTools

Command line-based work arounds

- if you just want to eliminate *identical* entries (here: region1 and region1Duplicate), use `sort` and `uniq` in the shell (note that the label of the identical regions is different - as `uniq` can only ignore fields at the beginning of a file, use `rev` to revert the sorted file, then `uniq` with ignoring the first field (which is now the name column) and then revert back:

```
$ sort -k1,1 -k2,2n testBed.bed | rev | uniq -f1 | rev
chr1        10       20       region1
chr1        7        15       region2
chr1        18       29       region3
chr1        35       40       region4
```

- if you would like to *merge all overlapping regions* into one big one, use the `mergeBed` from the BEDtools suite:
  - again, the BED file must be sorted first
  - `-n` and `-nms` tell `mergeBed` to output the number of overlapping regions and the names of them
  - in the resulting file, regions 1, 2 and 3 are merged

```
$ sort -k1,1 -k2,2n testBed.bed | mergeBed -i stdin -n -nms
chr1      7       29      region2;region1;region1Duplicate;region3        4
chr1      35      40      region4 1
```

- if you would like to *keep only regions that do not overlap* with any other region in the same BED file, use the same `mergeBed` routine but subsequently filter out those regions where several regions were merged.
  - the `awk` command will check the last field of each line (`$NF`) and will print the original line (`$0`) only if the last field contained a number smaller than 2

```
$ sort -k1,1 -k2,2n testBed.bed | mergeBed -i stdin -n -nms | awk '$NF < 2
↪{print $0}'
chr1      35      40      region4 1
```

1.7.8 Why does the maximum value in the heatmap not equal the maximum value in the matrix?

Additional processing, such as outlier removal, is done on the matrix prior to plotting the heatmap. We've found this beneficial in most cases. You can override this by manually setting `--zMax` and/or `` ` ``--zMin, respectively.

1.7.9 The heatmap I generated looks very "coarse", I would like a much more fine-grained image.

- decrease the *bin size* when generating the matrix using computeMatrix

In Galaxy:

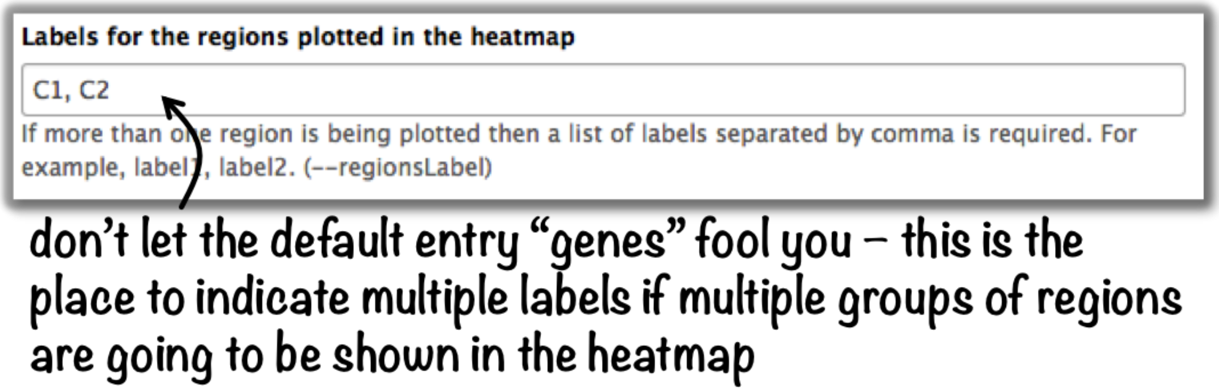
- go to "advanced options" –> "Length, in base pairs, of the non-overlapping *bin* for averaging the score over the regions length" –> define a smaller value, e.g. 50 or 25 bp
- make sure that you used a sufficiently small *bin* size when calculating the *bigWig* file, though (if generated with deepTools, you can check the option "bin size")

1.7.10 How can I change the automatic labels of the clusters in a k-means clustered heatmap?

Each cluster is treated exactly the same way as different groups of regions. Therefore, you can use the same option to define the labels of the final heatmap:

In Galaxy: plotHeatmap –> "Advanced output options" –> "Labels for the regions plotted in the heatmap".

If you indicated 2 clusters for k-means clustering, enter here: C1, C2, –> instead of the full default label ("cluster 1"), the heatmap will be labeled with the abbreviations.



Labels for the regions plotted in the heatmap

C1, C2

If more than one region is being plotted then a list of labels separated by comma is required. For example, label1, label2. (--regionsLabel)

don't let the default entry "genes" fool you – this is the place to indicate multiple labels if multiple groups of regions are going to be shown in the heatmap

In the command line, use the `--regionsLabel` option to define the customized names for the regions.

1.7.11 How can I manually specify several groups of regions (instead of clustering)?

Simply specify multiple BED files (e.g., genes.bed, exons.bed and introns.bed). This works both in Galaxy and on the command line.

1.7.12 What do I have to pay attention to when working with a draft version of a genome?

If your genome isn't included in our standard dataset then you'll need the following:

1. **Effective genome size** - this is mostly needed for bamCoverage and bamCompare, see *below* for details
2. **Reference genome sequence in 2bit format** - this is needed for computeGCBias, see *2bit* for details

---

1.7.13 How do I calculate the effective genome size for an organism that's not in your list?

At the moment we do not provide a tool for this purpose, so you'll have to find a solution outside of deepTools for the time being.

The "real" effective genome size is the part of the genome that is *uniquely mappable*. This means that the value will depend on the genome properties (how many repetitive elements, quality of the assembly etc.) and the length of the sequenced reads as 100 million 36-bp-reads might cover less than 100 million 100-bp-reads.

We currently have these options for you:

1. Use an *GEM*
2. Use *faCount* (only if you let reads be aligned non-uniquely, too!)
3. Use *bamCoverage*
4. Use *genomeCoverageBed*

Use `GEM`

There is a tool that promises to calculate the mappability for any genome given the read length (k-mer length): GEM-Mappability Calculator . According to this reply here, you can calculate the effective genome size after running this program by counting the numbers of "!" which stands for uniquely mappable regions.

Use `faCount`

If you are using `bowtie2`, which reports *multimappers* (i.e., *non-uniquely* mapped reads) as a default setting, you can use **faCount from UCSC tools** to report the total number of bases as well as the number of bases that are missing from the genome assembly indicated by 'N'. The effective genome size would then be the total number of base pairs minus the total number of 'N'. Here's an example output of `faCount` on *D. melanogaster* genome version dm3:

```
$ UCSCtools/faCount dm3.fa
#seq                len             A       C       G        T       N       cpg
chr2L               23011544        6699731 4811687 4815192  6684734 200     926264
chr2LHet    368872          90881   58504   57899    90588   71000   10958
chr2R               21146708        6007371 4576037 4574750  5988450 100     917644
chr2RHet    3288761         828553  537840   529242  826306  566820  99227
chr3L               24543557        7113242 5153576  5141498 7135141 100     995078
chr3LHet    2555491         725986  473888   479000  737434 139183   89647
chr3R               27905053        7979156 5995211  5980227 7950459 0       1186894
chr3RHet    2517507         678829  447155   446597  691725  253201  84175
chr4                1351857         430227  238155   242039  441336  100     43274
chrU                10049037        2511952 1672330  1672987 2510979 1680789 335241
chrUextra   29004656        7732998 5109465  5084891 7614402 3462900 986216
chrX                22422827        6409325 4742952  4748415 6432035 90100   959534
chrXHet             204112          61961   40017    41813   60321  0        754
chrYHet             347038          74566   45769    47582   74889  104232   8441
chrM                19517           8152    2003     1479    7883   0        132
total               168736537       47352930 33904589 33863611 47246682 6368725␣
 ↪6650479
```

In this example: Total no. bp = 168,736,537 Total no. 'N' = 6,368,725

Warning: This method only works if multimappers are randomly assigned to their possible locations (in such cases the effective genome size is simply the number of non-N bases).

Use `bamCoverage`

If you have a sample where you expect the genome to be covered completely, e.g. from genome sequencing, a very trivial solution is to use tools/bamCoverage with a bin size of 1 bp and the `--outFileFormat` option set to 'bedgraph'. You can then count the number of non-Zero bins (bases) which will indicate the mappable genome size for this specific sample.

Use `genomeCoverageBed`

`genomeCoverageBed` from the BEDtools suite can be used to calculate the number of bases in the genome for which 0 overlapping reads can be found. As described on the BEDtools website (go to genomeCov description), you need:

- a file with the chromosome sizes of your sample's organism

- a position-sorted BAM file

```
$ bedtools genomecov -ibam sortedBAMfile.bam -g genome.size
```

1.7.14 Where can I download the 2bit genome files required for `computeGCBias`?

The 2bit files of most genomes can be found here. Search for the .2bit ending. Otherwise, **fasta files can be converted to 2bit** using the UCSC program faToTwoBit (available for different platforms from UCSC here).

deepTools Galaxy.	code @ github.

1.8 Galaxy-related FAQ

- *I've reached my quota - what can I do to save some space?*
- *Copying from one history to another doesn't work for me - the data set simply doesn't show up in the target history!*
- *How can I use a published workflow?*
- *I would like to use one of your workflows - not in the deepTools Galaxy, but in the local Galaxy instance provided by my institute. Is that possible?*
- `plotProfile` *says that one option will only work if "computeMatrix was run with –missingDataAsZero". How can I find out whether I ran* `computeMatrix` *that way?*
- *How can I have a look at the continuous read coverages from bigWig files? Which genome browser do you recommend?*
  - *IGV (recommended)*
  - *UCSC*
- *What's the best way to integrate the deepTools results with other downstream analyses (outside of Galaxy)?*
- *How can I determine basic parameters of a BAM file, such as the number of reads, read length, duplication rate and average DNA fragment length?*

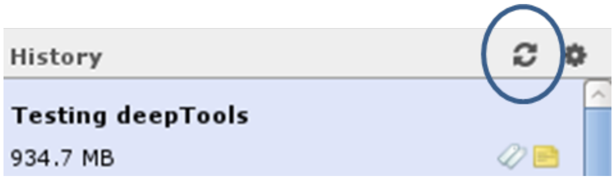
1.8.1 I've reached my quota - what can I do to save some space?

1. make sure that all the data sets you deleted are **permanently** eliminated from our disks: go to the history option button and select "Purge deleted data sets", then hit the "refresh" button on top of your history panel
2. download all data sets for which you've completed the analysis, then remove the data sets (click on the "x" and then **make sure they're purged** (see above)).

1.8.2 Copying from one history to another doesn't work for me - the data set simply doesn't show up in the target history!

Once you've copied a data set from one history to another, check two things:

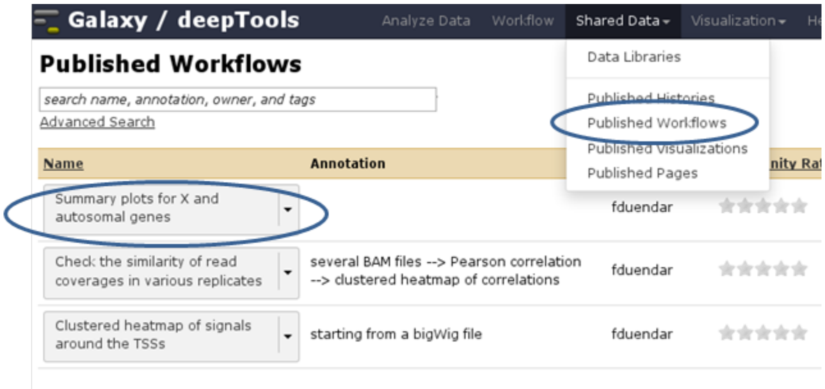
- do you see the destination history in your history panel, i.e. does the title of the current history panel match the name of the destination history you selected in the main frame?
- hit the refresh button


History
Testing deepTools
934.7 MB

1.8.3 How can I use a published workflow?

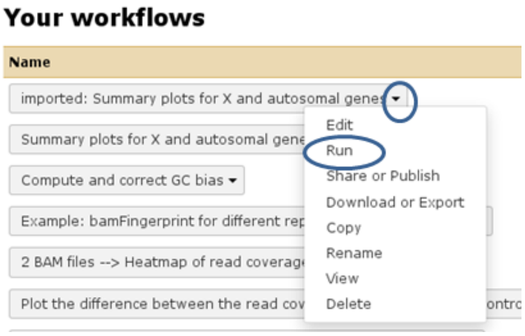
You **must register** if you want to use the workflows within deepTools Galaxy. ("User" –> "Register" - all you have to supply is an email address). Make sure to read the Terms of Use, though!

You can find workflows that are public or specifically shared with you by another user via "Shared Data" –> "Published Workflows". Click on the triangle next to the workflow you're interested in and select "import".


Galaxy / deepTools
Analyze Data
Workflow
Shared Data
Visualization
He
Published Workflows
search name, annotation, owner, and tags
Advanced Search
Data Libraries
Published Histories
Published Workflows
Published Visualizations
Published Pages

Name	Annotation		nity Rat
Summary plots for X and autosomal genes		fduendar	★★★★★
Check the similarity of read coverages in various replicates	several BAM files --> Pearson correlation --> clustered heatmap of correlations	fduendar	★★★★★
Clustered heatmap of signals around the TSSs	starting from a bigWig file	fduendar	★★★★★

A green box should appear, there you select "start using this workflow", which should lead you to your own workflow menu (that you can always access via the top menu "Workflow"). Here, you should now see a workflow labeled "imported: . . . ". If you want to use the workflow right away, click on the triangle and select "Run". The workflow should now be available within the Galaxy main data frame and should be waiting for your input.


Your workflows
Name
imported: Summary plots for X and autosomal genes
Summary plots for X and autosomal gene
Compute and correct GC bias
Example: bamFingerprint for different rep
2 BAM files --> Heatmap of read coverag
Plot the difference between the read cov
ontro
Edit
Run
Share or Publish
Download or Export
Copy
Rename
View
Delete

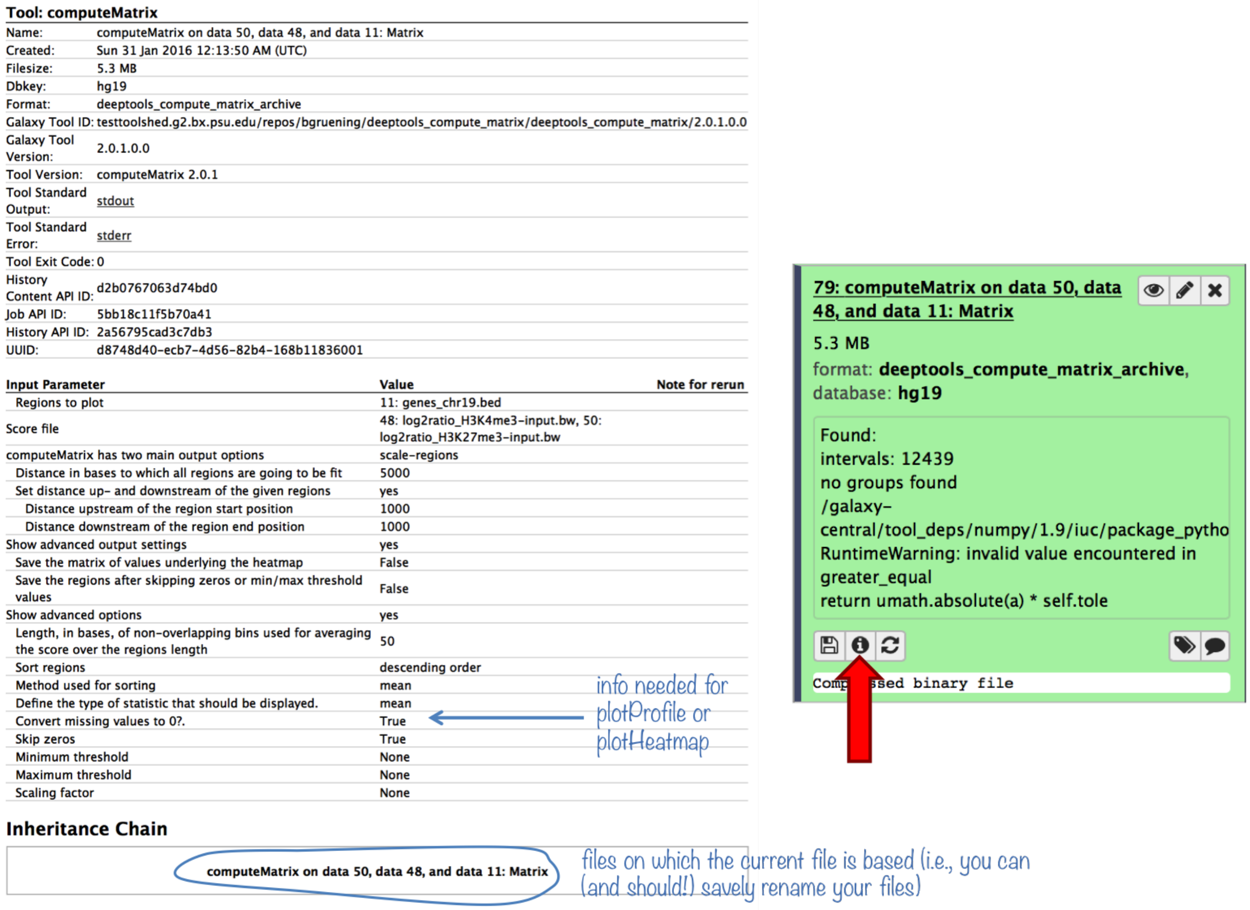
1.8.4 I would like to use one of your workflows - not in the deepTools Galaxy, but in the local Galaxy instance provided by my institute. Is that possible?

Yes, it is possible. The only requirement is that your local Galaxy has a recent installation of deepTools.

Go to the workflows, click on the ones you're interested in and go to "Download". This will save the workflows into .ga files on your computer. Now go to your local Galaxy installation and login. Go to the workflow menu and select "import workflow" (top right hand corner of the page). Click on "Browse" and select the saved workflow. If you have the same tool versions installed in your local Galaxy, these workflows should work right away.

1.8.5 `plotProfile` says that one option will only work if "computeMatrix was run with –missingDataAsZero". How can I find out whether I ran `computeMatrix` that way?

Galaxy keeps track of everything you do. To see which options you chose to generate a specific data set, simply click on the "info" button.



Tool: computeMatrix

Name:	computeMatrix on data 50, data 48, and data 11: Matrix
Created:	Sun 31 Jan 2016 12:13:50 AM (UTC)
Filesize:	5.3 MB
Dbkey:	hg19
Format:	deeptools_compute_matrix_archive
Galaxy Tool ID:	testtoolshed.g2.bx.psu.edu/repos/bgruening/deeptools_compute_matrix/deeptools_compute_matrix/2.0.1.0.0
Galaxy Tool Version:	2.0.1.0.0
Tool Version:	computeMatrix 2.0.1
Tool Standard Output:	stdout
Tool Standard Error:	stderr
Tool Exit Code:	0
History Content API ID:	d2b0767063d74bd0
Job API ID:	5bb18c11f5b70a41
History API ID:	2a56795cad3c7db3
UUID:	d8748d40-ecb7-4d56-82b4-168b11836001

Input Parameter	Value	Note for rerun
Regions to plot	11: genes_chr19.bed	
Score file	48: log2ratio_H3K4me3-input.bw, 50: log2ratio_H3K27me3-input.bw	
computeMatrix has two main output options	scale-regions	
Distance in bases to which all regions are going to be fit	5000	
Set distance up- and downstream of the given regions	yes	
Distance upstream of the region start position	1000	
Distance downstream of the region end position	1000	
Show advanced output settings	yes	
Save the matrix of values underlying the heatmap	False	
Save the regions after skipping zeros or min/max threshold values	False	
Show advanced options	yes	
Length, in bases, of non-overlapping bins used for averaging the score over the regions length	50	
Sort regions	descending order	
Method used for sorting	mean	
Define the type of statistic that should be displayed.	mean	
Convert missing values to 0?.	True	
Skip zeros	True	
Minimum threshold	None	
Maximum threshold	None	
Scaling factor	None	

info needed for plotProfile or plotHeatmap

Inheritance Chain

computeMatrix on data 50, data 48, and data 11: Matrix

files on which the current file is based (i.e., you can (and should!) savely rename your files)

79: computeMatrix on data 50, data 48, and data 11: Matrix

5.3 MB

format: **deeptools_compute_matrix_archive**, database: **hg19**

Found:
intervals: 12439
no groups found
/galaxy-central/tool_deps/numpy/1.9/iuc/package_pytho
RuntimeWarning: invalid value encountered in greater_equal
return umath.absolute(a) * self.tole

`Comp ssed binary file`

1.8.6 How can I have a look at the continuous read coverages from bigWig files? Which genome browser do you recommend?

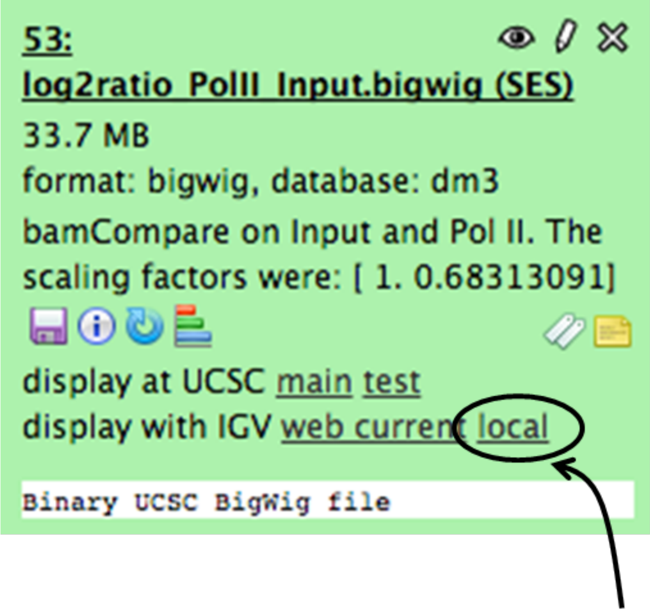
There are 2 popular genome browsers for visualizing continuous data: UCSC and IGV.

IGV (recommended)

We recommend downloading IGV, which is free for academic use. IGV itself needs an up-to-date Java installation and a considerable amount of RAM. It's usage is rather intuitive and the display can be easily customized. In addition, you can download genome-wide annotation data that can be displayed together with your own data.

To display data in IGV, do the following:

1. Go to http://www.broadinstitute.org/igv/, register and download IGV
2. Unpack the IGV archive and change to the extracted IGV folder
3. Use the `igv.bat` (Windows), `igv.sh` (Linux) or `igv.command` (OSX) to start IGV (for more information please read the included `readme.txt` file or the IGV documentation).
4. Choose the genome version of the file(s) you would like to visualize (e.g. dm3) **THIS IS THE MOST IMPORTANT STEP!** IGV will not detect the genome version automatically, i.e. if you select mm9 but your file is based on human data, it will still be displayed without an error message (but with the wrong positions, obviously!)
5. Go to your deepTools Galaxy server (http://deeptools.ie-freiburg.mpg.de/) and navigate to your data set of choice
6. Click on your data set so that you see its details like in the screenshot below (**Keep in mind that not all datasets can be visualized in IGV or UCSC.** We recommend to use *bigWig* or *BED* files for visualization.)



53:
log2ratio PolII Input.bigwig (SES)
33.7 MB
format: bigwig, database: dm3
bamCompare on Input and Pol II. The scaling factors were: [1. 0.68313091]
display at UCSC main test
display with IGV web current local

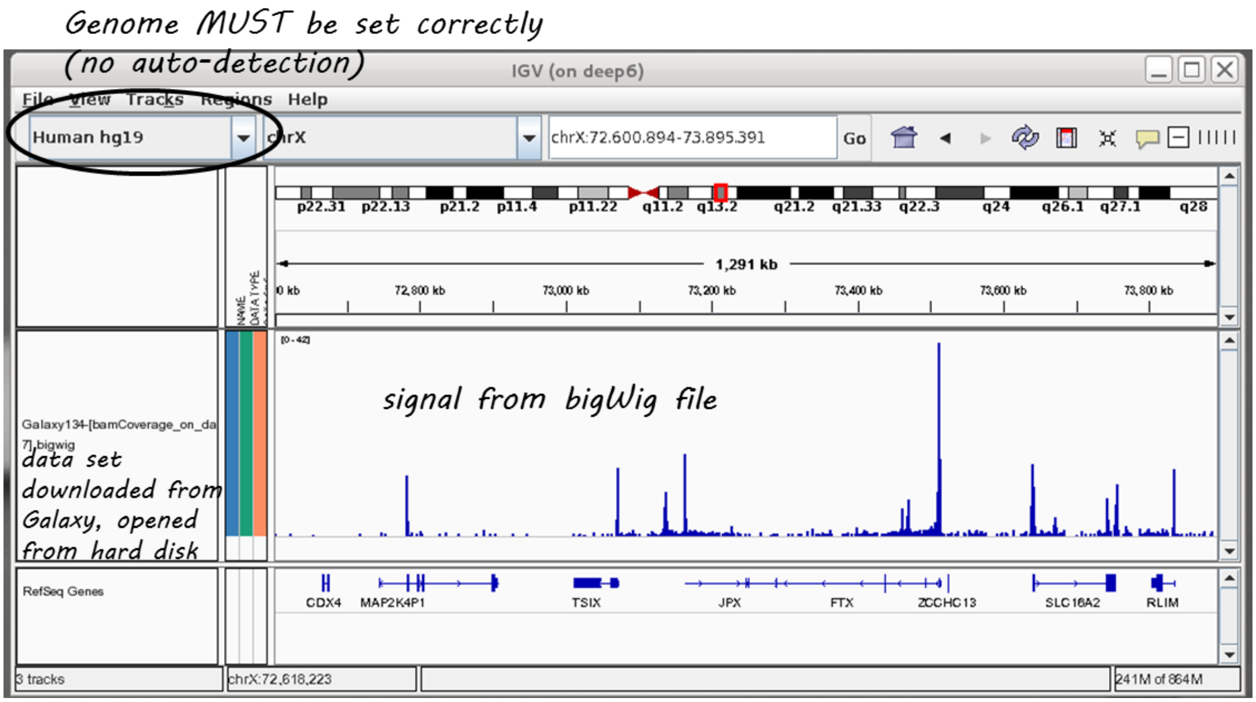
Binary UCSC BigWig file

IGV should already be running on your computer BEFORE clicking here

Now click on **"display with IGV local"** to visualize your data set in IGV that should already be running on your computer.

Note: "display with IGV Web current" can be used if you do not have an installed IGV. It will start an IGV web start version. **We do *not* recommend that option.**

Here’s a screenshot of a typical bigWig file display:


IGV (on deep6)
File View Tracks Regions Help
Human hg19
chrX
chrX:72.600.894-73.895.391
Go
p22.31 p22.13 p21.2 p11.4 p11.22 q11.2 q13.2 q21.2 q21.33 q22.3 q24 q26.1 q27.1 q28
1,291 kb
0 kb 72,800 kb 73,000 kb 73,200 kb 73,400 kb 73,600 kb 73,800 kb
NAME
DATA TYPE
[0 - 42]
Galaxy134-[bamCoverage_on_da
7].bigwig
RefSeq Genes
CDX4 MAP2K4P1 TSIX JPX FTX ZCCHC13 SLC16A2 RLIM
3 tracks
chrX:72,618,223
241M of 864M

Genome MUST be set correctly (no auto-detection)

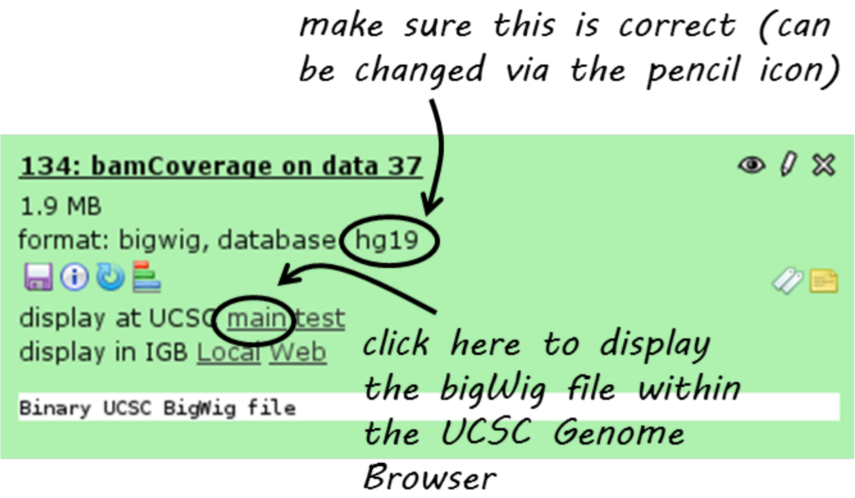
signal from bigWig file

data set downloaded from Galaxy, opened from hard disk

For more information, check out the IGV documentation.

UCSC

There is a direct link from within deepTools Galaxy to stream a data set to UCSC. You can find it in the data set tiles: “display at UCSC”, like here:



make sure this is correct (can be changed via the pencil icon)

134: bamCoverage on data 37

1.9 MB

format: bigwig, database hg19

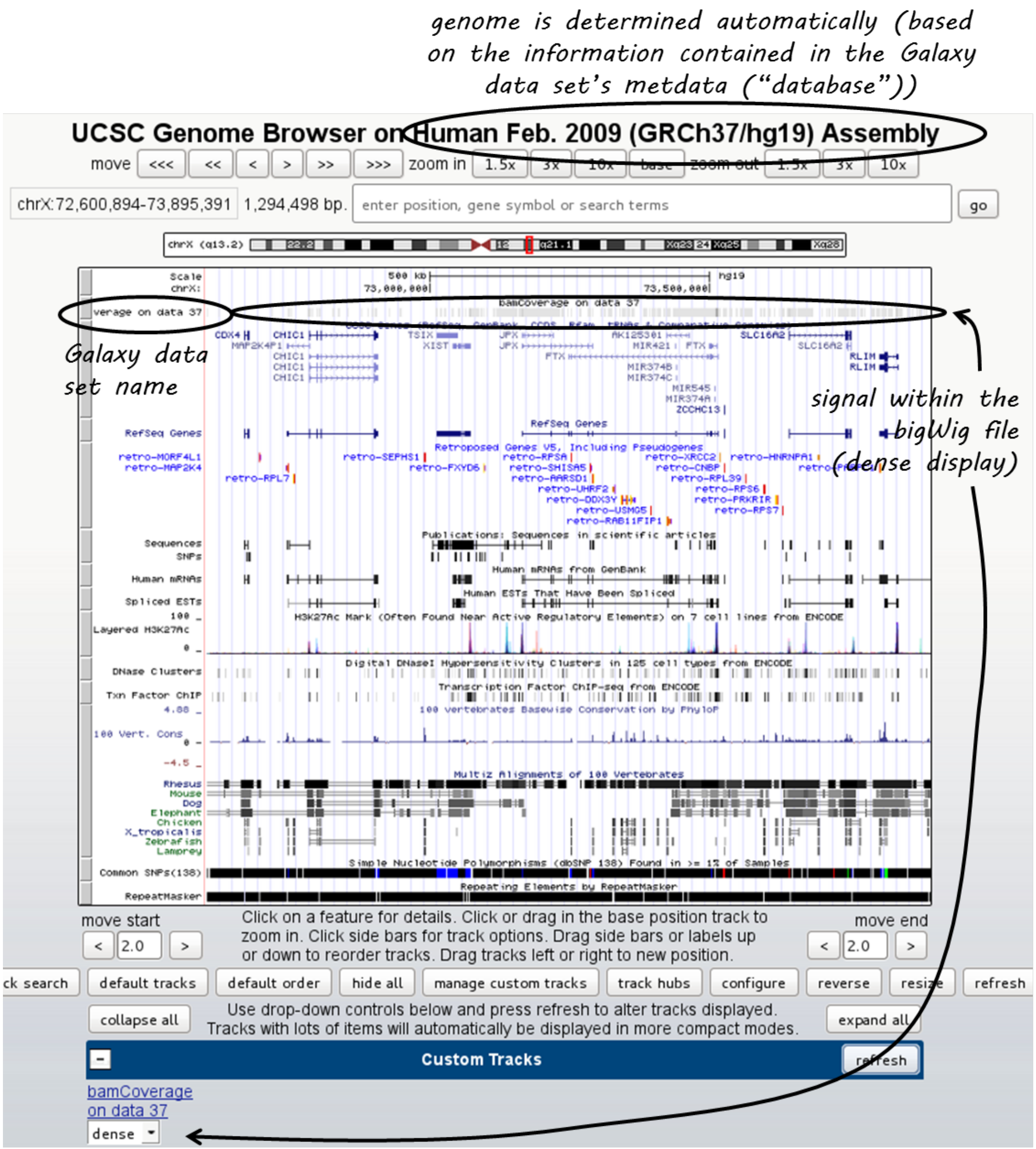
display at UCSC main test

display in IGB Local Web

Binary UCSC BigWig file

click here to display the bigWig file within the UCSC Genome Browser

Click on “main” and the UCSC browser should open within a new window, displaying the data set that you chose. The default setting for bigWig files is the “dense” display that looks like a heatmap.



genome is determined automatically (based on the information contained in the Galaxy data set's metdata ("database"))

Galaxy data set name

signal within the bigWig file (dense display)

UCSC Genome Browser on Human Feb. 2009 (GRCh37/hg19) Assembly
move <<< << < > >> >>> zoom in 1.5x 3x 10x base zoom out 1.5x 3x 10x
chrX:72,600,894-73,895,391 1,294,498 bp.
enter position, gene symbol or search terms
go
bamCoverage on data 37
RefSeq Genes
Layered H3K27Ac
DNase Clusters
Txn Factor ChIP
100 Vert. Cons
Common SNPs(138)
RepeatMasker
move start
move end
ck search
default tracks
default order
hide all
manage custom tracks
track hubs
configure
reverse
resize
refresh
collapse all
expand all
Custom Tracks
refresh
bamCoverage on data 37
dense

Click on a feature for details. Click or drag in the base position track to zoom in. Click side bars for track options. Drag side bars or labels up or down to reorder tracks. Drag tracks left or right to new position.

Use drop-down controls below and press refresh to alter tracks displayed. Tracks with lots of items will automatically be displayed in more compact modes.

If you would like to display the continuous profile in a "valley-mountain" fashion like the one shown in the IGV screenshot, go to the drop-down menu underneath your custom track and choose "full".

UCSC has large amounts of public data that you can display which you can find by scrolling down the page, beyond your custom track entry. For more information on how to use the UCSC Genome Browser, go here.

Known issues with UCSC

- **chromosome naming**: UCSC expects chromosome names to be indicated in the format "chr"Number, e.g. chr1. If you mapped your reads to a non-UCSC-standard genome, chances are that chromosomes are labeled just with their number. bigWig files generated from these BAM files will not be recognized by UCSC, i.e. you will see the data set name, but no signal.

- **no upload of bigWig files from your hard drive**: to minimize the computational strains, UCSC relies on streaming bigWig files (i.e. there's no need to load the entire file at once, the browser will always just load the data for the specific region a user is looking at).

1.8.7 What's the best way to integrate the deepTools results with other downstream analyses (outside of Galaxy)?

You can **save all the data tables** underlying every image produced by deepTools, i.e. if you would like to plot the average profiles in a different way, you could download the corresponding data (after ticking the relevant option under "advanced output options") and import them into R, Excel, GraphPadPrism etc.

The descriptions of the tools within Galaxy will also contain details on how to save the data and what sort of format to expect.

1.8.8 How can I determine basic parameters of a BAM file, such as the number of reads, read length, duplication rate and average DNA fragment length?

If you downloaded the *BAM* file from a public repository, chances are that those characteristics are in fact noted there.

If that's not the case, we recommend to have a look at the tool FastQC, which will return all of the above points (except the fragment size). The fragment size distribution can be obtained using the deepTools' tools/bamPEFragmentSize (since deepTools 2.0).

deepTools Galaxy.	code @ github.

1.9 Glossary of NGS terms

Like most specialized fields, next-generation sequencing has inspired many an acronyms. We are trying to keep track of those *Abbreviations* that we heavily use. Do make us aware if something is unclear by opening an issue on github

- *Abbreviations*
- *NGS and generic terminology*
  - *bin*
  - *Input*
  - *read*
- *File Formats*
  - *2bit*
  - *BAM*
  - *BED*

  - *bedGraph*
  - *bigWig*
  - *FASTA*
  - *FASTQ*
  - *SAM*
    - *SAM header section*
    - *SAM alignment section*

1.9.1 Abbreviations

Reference genomes are usually referred to by their abbreviations, such as:

- hg19 = human genome, version 19
- mm9 = *Mus musculus* genome, version 9
- dm3 = *Drosophila melanogaster*, version 3
- ce10 = *Caenorhabditis elegans*, version 10

For a more comprehensive list of available reference genomes and their abbreviations, see the UCSC data base.

Acronym	full phrase	Synonyms/Explanation
<ANYTHING>-seq	sequencing	indicates that an experiment was completed by DNA sequencing using NGS
ChIP-seq	chromatin immunoprecipitation sequencing	NGS technique for detecting transcription factor binding sites and histone modifications (see entry *Input* for more information)
DNase	deoxyribonuclease I	DNase I digestion is used to determine active ("open") chromatin regions
HTS	high-throughput sequencing	next-generation sequencing, massive parallel short read sequencing, deep sequencing
MNase	micrococcal nuclease	MNase digestion is used to determine sites with nucleosomes
NGS	next-generation sequencing	high-throughput (DNA) sequencing, massive parallel short read sequencing, deep sequencing
RPGC	reads per genomic content	normalize reads to 1x sequencing depth, sequencing depth is defined as: (mapped reads x fragment length) / effective genome size
RPKM	reads per kilobase per million reads	normalize read numbers: RPKM (per bin) = reads per bin / (mapped reads (in millions) x bin length (kb))

For a review of popular *-seq applications, see Zentner and Henikoff.

1.9.2 NGS and generic terminology

The following are terms that may be new to some:

bin

- synonyms: window, region
- A 'bin' is a subset of a larger grouping. Many calculations calculation are performed by first dividing the genome into small regions (bins), on which the calculations are actually performed.

Input

- Control experiment typically done for ChIP-seq experiments
- While ChIP-seq relies on antibodies to enrich for DNA fragments bound to a certain protein, the input sample should be processed exactly the same way, excluding the antibody. This allows one to account for biases introduced by sample handling and the general chromatin structure of the cells

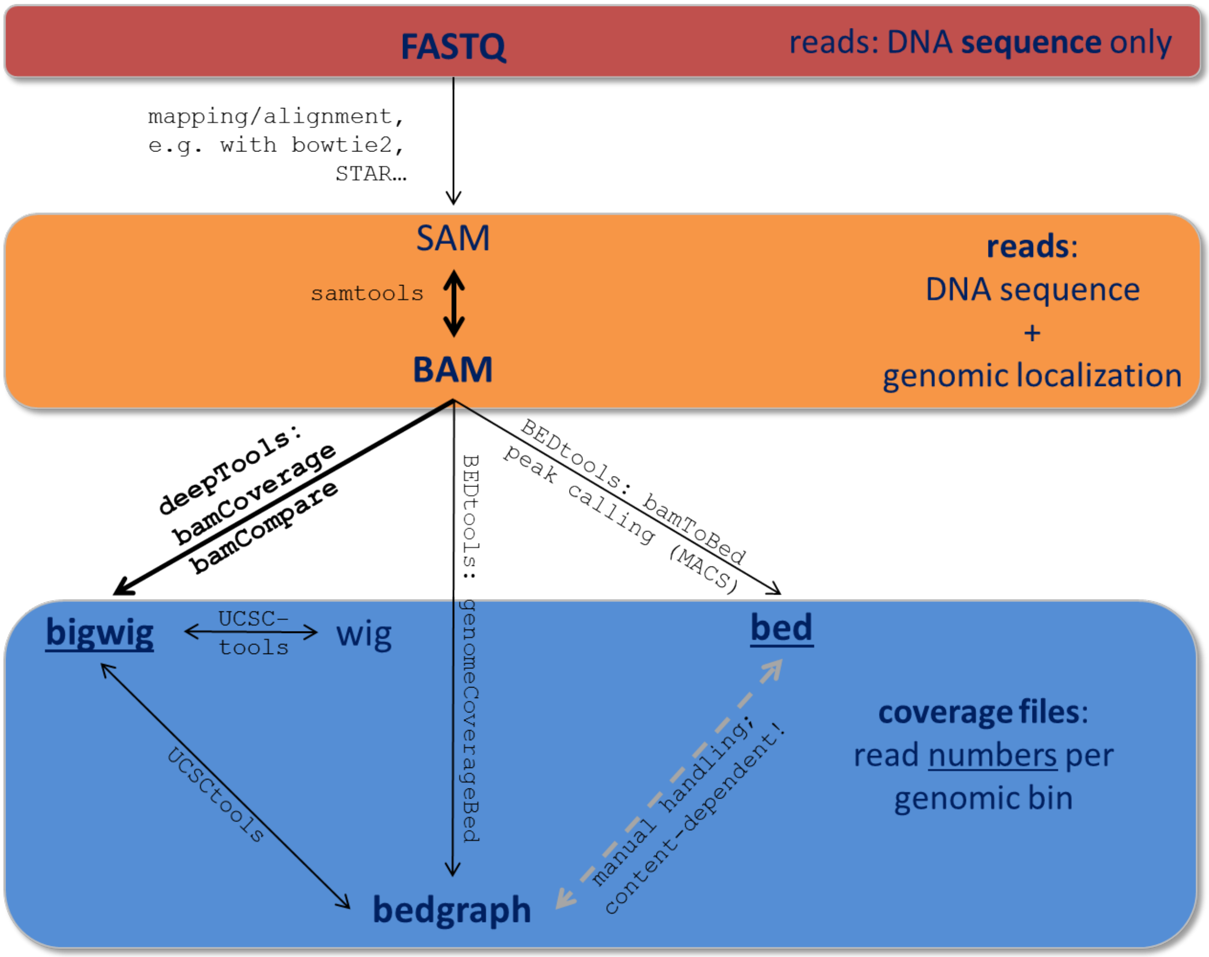
read

- synonym: tag
- This term refers to the piece of DNA that is sequenced ("read") by the sequencers. We try to differentiate between "read" and "DNA fragment" as the fragments that are put into the sequencer tend to be in the range of 200-1000 bases, of which only the first 50 to 300 bases are typically sequenced. Most of the deepTools will not only take these reads into account, but also extend them to match the original DNA fragment size. (The original size will either be given by you or, if you used paired-end sequencing, be calculated from the distance between the two read mates).

1.9.3 File Formats

Data obtained from next-generation sequencing data must be processed several times. Most of the processing steps are aimed at extracting only that information needed for a specific down-stream analysis, with redundant entries often discarded. Therefore, **specific data formats are often associated with different steps of a data processing pipeline**.

Here, we just want to give very brief key descriptions of the file, for elaborate information we will link to external websites. Be aware, that the file name sorting here is alphabetical, not according to their usage within an analysis pipeline that is depicted here:


FASTQ
reads: DNA sequence only
mapping/alignment,
e.g. with bowtie2,
STAR...
SAM
samtools
BAM
reads:
DNA sequence
+
genomic localization
deepTools:
bamCoverage
bamCompare
BEDtools: genomeCoverageBed
BEDtools: bamToBed
peak calling (MACS)
bigwig
UCSC-
tools
wig
bed
UCSCtools
manual handling;
content-dependent!
bedgraph
coverage files:
read numbers per
genomic bin

Follow the links for more information on the different tool collections mentioned in the figure:

samtools | UCSCtools | BEDtools |

2bit

- compressed, binary version of genome sequences that are often stored in *FASTA*
- most genomes in 2bit format can be found at UCSC
- *FASTA* files can be converted to 2bit using the UCSC programm *faToTwoBit*, which is available for different platforms at UCSC
- more information can be found here

BAM

- typical file extension: `.bam`
- *binary* file format (complement to *SAM*)
- contains information about sequenced reads (typically) *after alignment* to a reference genome
- **each line = 1 mapped read, with information about:**
    - its mapping quality (how likelihood that the reported alignment is correct)

  - its sequencing quality (the probability that each base is correct)
  - its sequence
  - its location in the genome
  - etc.
- highly recommended format for storing data
- to make a BAM file human-readable, one can, for example, use the program *samtools view*
- for more information, see below for the definition of *SAM* files

BED

- typical file extension: `.bed`
- text file
- used for genomic intervals, e.g. genes, peak regions etc.
- the format can be found at UCSC
- for deepTools, the first 3 columns are important: chromosome, start position of the region, end position of the genome
- do not confuse it with the *bedGraph* format (although they are related)
- example lines from a BED file of mouse genes (note that the start position is 0-based, the end-position 1-based, following UCSC conventions for BED files):

```
chr1    3204562 3661579 NM_001011874 Xkr4   -
chr1    4481008 4486494 NM_011441    Sox17  -
chr1    4763278 4775807 NM_001177658 Mrpl15 -
chr1    4797973 4836816 NM_008866    Lypla1 +
```

bedGraph

- typical file extension: `.bg`, `.bedGraph`
- text file
- similar to BED file (not the same!), it can *only* contain 4 columns and the 4th column *must* be a score
- again, read the UCSC description for more details
- 4 example lines from a bedGraph file (like BED files following the UCSC convention, the start position is 0-based, the end-position 1-based in bedGraph files):

```
chr1 10 20 1.5
chr1 20 30 1.7
chr1 30 40 2.0
chr1 40 50 1.8
```

bigWig

- typical file extension: `.bw`, `.bigwig`
- *binary* version of a *bedGraph* or `wig` file

- contains coordinates for an interval and an associated score
- the score can be anything, e.g. an average read coverage
- UCSC description for more details

FASTA

- typical file extension: `.fasta`
- text file, often gzipped (`.fasta.gz`)
- very simple format for **DNA/RNA** or **protein** sequences, this can be anything from small pieces of DNA or proteins to an entire genome (most likely, you will get the genome sequence of your organism of interest in fasta format)
- see the *2bit* file format entry for a compressed alternative
- example from wikipedia showing exactly one sequence:

```
>gi|5524211|gb|AAD44166.1| cytochrome b [Elephas maximus maximus]
 LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFWGATVITNLFSAIPYIGTNLV
 EWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKDFLG
 LLILILLLLLLALLSPDMLGDPDNHMPADPLNTPLHIKPEWYFLFAYAILRSVPNKLGGVLALFLSIVIL
 GLMPFLHTSKHRSMMLRPLSQALFWTLTMDLLTLTWIGSQPVEYPYTIIGQMASILYFSIILAFLPIAGX
 IENY
```

FASTQ

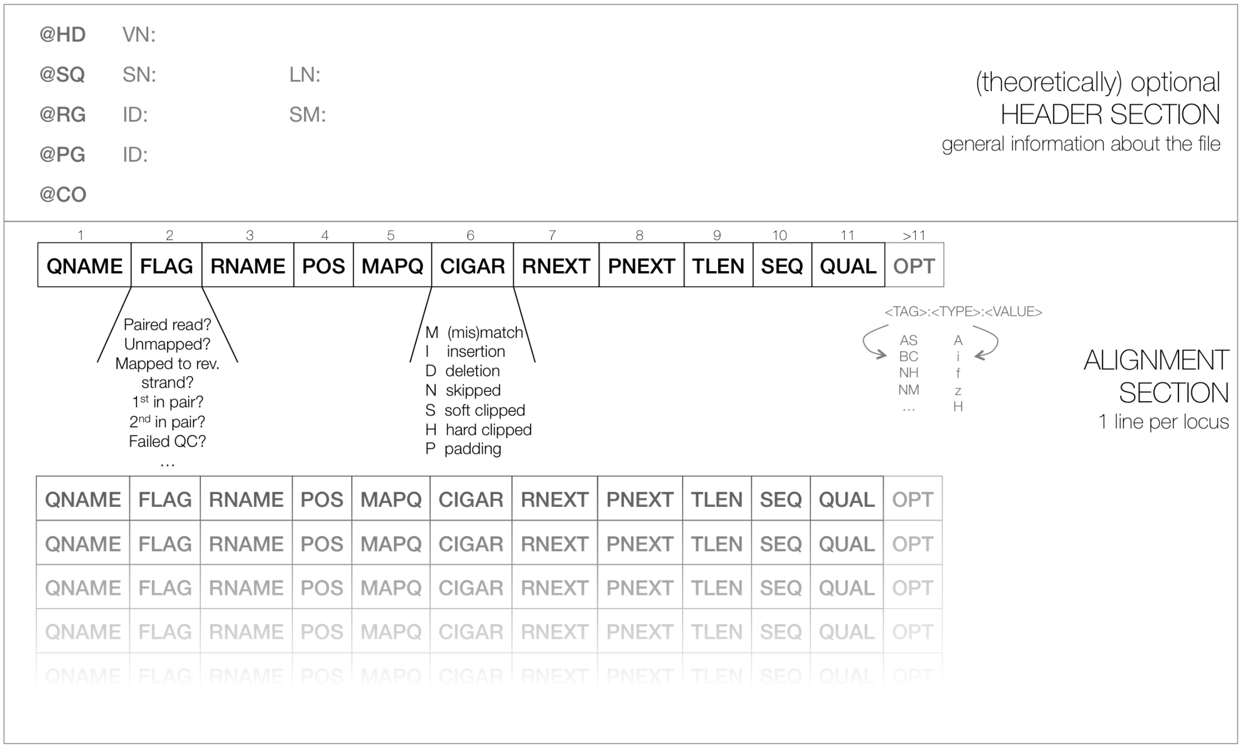
- typical file extension: `.fastq`, `.fq`
- text file, often gzipped (–> `.fastq.gz`)
- **contains raw read information – 4 lines per read:**
  - read ID
  - base calls
  - additional information or empty line
  - sequencing quality measures - 1 per base call
- note that there is no information about where in the genome the read originated from
- example from the wikipedia page, which contains further information:

```
@read001                                                                                                ␣
↪                                   # read ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT        # read␣
↪sequence
+                                                                                                       ␣
↪                                           # usually empty line
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65        # ASCII-
↪encoded quality scores
```

- if you need to find out what type of ASCII-encoding your .fastq file contains, you can simply run FastQC – its summery file will tell you

SAM

- typical file extension: `.sam`
- usually the result of an alignment of sequenced reads to a reference genome
- contains a short header section (entries are marked by @ signs) and an alignment section where each line corresponds to a single read (thus, there can be millions of these lines)


@HD VN:
@SQ SN: LN:
@RG ID: SM:
@PG ID:
@CO
(theoretically) optional
HEADER SECTION
general information about the file
1 2 3 4 5 6 7 8 9 10 11 >11
QNAME FLAG RNAME POS MAPQ CIGAR RNEXT PNEXT TLEN SEQ QUAL OPT
Paired read?
Unmapped?
Mapped to rev. strand?
1st in pair?
2nd in pair?
Failed QC?
…
M (mis)match
I insertion
D deletion
N skipped
S soft clipped
H hard clipped
P padding
<TAG>:<TYPE>:<VALUE>
AS BC NH NM …
A i f z H
ALIGNMENT SECTION
1 line per locus

SAM header section

- tab-delimited lines, beginning with @, followed by tag:value pairs
- *tag* = two-letter string that defines the content and the format of *value*

SAM alignment section

- each line contains information about its mapping quality, its sequence, its location in the genome etc.

```
r001 163 chr1 7 30 8M2I4M1D3M = 37 39 TTAGATAAAGGATACTG *
r002 0 chr1 9 30 3S6M1P1I4M * 0 0 AAAAGATAAGGATA *
```

- the **flag in the second field** contains the answer to several yes/no assessments that are encoded in a single number
- for more details on the flag, see this thorough explanation or this more technical explanation
- the **CIGAR string in the 6th field** represents the types of operations that were needed in order to align the read to the specific genome location:
    - insertion

  - deletion (small deletions denoted with *D*, bigger deletions, e.g., for spliced reads, denoted with *N*)
  - clipping (deletion at the ends of a read)

> **Warning:** Although the SAM/BAM format is rather meticulously defined and documented, whether an alignment program will produce a SAM/BAM file that adheres to these principles is completely up to the programmer. The mapping score, CIGAR string, and particularly, **all optional flags** (fields >11) are often **very differently defined depending on the program**. If you plan on filtering your data based on any of these criteria, make sure you know exactly how these entries were calculated and set!

deepTools Galaxy.	code @ github.

1.10 deepTools API

deepTools consists of several command line and Galaxy wrappers for summarizing the information of Next Generation Sequencing data that can be mapped to a reference genome. Through the API, the engine powering the deepTools commands can be used for other purposes as well.

Our *deepTools API example* explains step-by-step how to make use of some deepTools modules to achieve analyses outside the scope of the deepTools suite such as counting reads for certain genome regions and computing the FRiP score.

1.10.1 deepTools API example

The following is a short overview of the most useful methods and classes from deepTools. Complete information can be found in the following links: genindex and modindex

Finding read coverage over a region

With deepTools, the read coverage over multiple genomic regions and multiple files can be computed quite quickly using multiple processors. First, we start with a simple example that is later expanded upon to demonstrate the use of multipe processors. In this example we compute the coverage of reads over a small region for bins of 50bp. For this we need the *`deeptools.countReadsPerBin`* class.

```
import deeptools.countReadsPerBin as crpb
```

We also need a BAM file containing the aligned reads. The BAM file must be indexed to allow quick access to reads falling into the regions of interest.

```
bam_file = "file.bam"
```

Now, the `CountReadsPerBin` object can be initialized. The first argument to the constructor is a list of BAM files, which in this case is just one file. We are going to use a `binLength` of 50 bases, with subsequent bins adjacent (i.e., the `stepSize` between bins is also 50 bases). Overlapping bin coverages can be used by setting a `stepSize` smaller than `binLength`.

```
cr = crpb.CountReadsPerBin([bam_file], binLength=50, stepSize=50)
```

Now, we can compute the coverage over a region in chromosome 2 from position 0 to 1000.

```
cr.count_reads_in_region('chr2L', 0, 1000)
```

```
(array([[ 2.],
        [ 3.],
        [ 1.],
        [ 2.],
        [ 3.],
        [ 2.],
        [ 4.],
        [ 3.],
        [ 2.],
        [ 3.],
        [ 4.],
        [ 6.],
        [ 4.],
        [ 2.],
        [ 2.],
        [ 1.]]), '')
```

The result is a tuple with the first element a numpy array with one row per bin and one column per bam file. Since only one BAM file was used, there is only one column. If a file name for saving the raw data had been specificied, then the temporary file name used for this would appear in the second item of the tuple.

Filtering reads

If reads should be filtered, the relevant options simply need to be passed to the constructor. In the following code, the reads are filtered such that only those with a mapping quality of at least 20 and not aligned to the reverse strand are kept (samFlag_exclude=16, where 16 is the value for reverse reads, see the [SAM Flag Calculator](http://broadinstitute.github.io/picard/explain-flags.html) for more info). Furthermore, duplicated reads are ignored.

```
cr = crpb.CountReadsPerBin([bam_file], binLength=50, stepSize=50,
                           minMappingQuality=20,
                           samFlag_exclude=16,
                           ignoreDuplicates=True
                           )
cr.count_reads_in_region('chr2L', 1000000, 1001000)
```

```
(array([[ 1.],
        [ 1.],
        [ 0.],
        [ 0.],
        [ 0.],
        [ 0.],
        [ 2.],
        [ 3.],
        [ 1.],
        [ 0.],
        [ 1.],
        [ 2.],
        [ 0.],
        [ 0.],
        [ 1.],
        [ 2.],
        [ 1.],
        [ 0.],
```

(continues on next page)

(continued from previous page)

```
        [ 0.],
        [ 0.]]), '')
```

Sampling the genome

Instead of adjacent bins, as in the previous cases, a genome can simply be sampled. This is useful to estimate some values, like depth of sequencing, without having to look at the complete genome. In the following example, 10,000 positions of size 1 base are going to be queried from three bam files to compute the average depth of sequencing. For this, we set the *numberOfSamples* parameter in the object constructor.

The *run()* method is used instead of *count_reads_in_region* to provide efficient sampling over the entire genome.

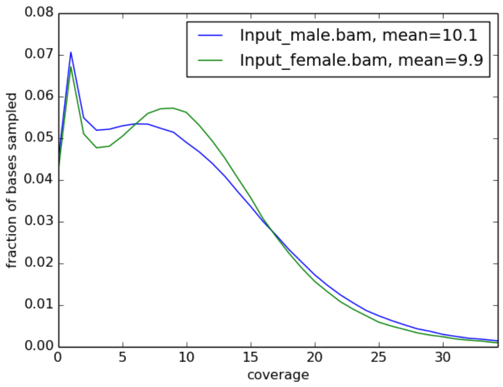
```
cr = crpb.CountReadsPerBin([bam_file1, bam_file2, bam_file3],
                           binLength=1, numberOfSamples=10000,
                           numberOfProcessors=10)
sequencing_depth = cr.run()
print sequencing_depth.mean(axis=0)
```

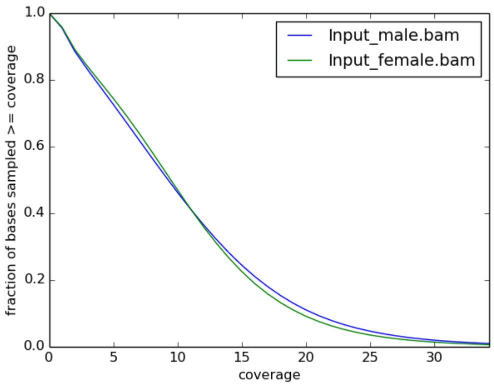
```
[  1.98923924   2.43743744  22.90102603]
```

The *run()* method splits the computation over 10 processors and collates the results. When the parameter *numberOfSamples* is used, the regions selected for the computation of the coverage are not random. Instead, the genome is split into 'number-of-samples' equal parts and the start of each part is queried for its coverage. You can also compute coverage over selected regions by inputting a BED file.

Now it is possible to make some diagnostic plots from the results:

```
fig, axs = plt.subplots(1, 2, figsize=(15,5))
# plot coverage
for col in res.T:
    axs[0].plot(np.bincount(col.astype(int)).astype(float)/total_sites)
    csum = np.bincount(col.astype(int))[::-1].cumsum()
    axs[1].plot(csum.astype(float)[::-1] / csum.max())
axs[0].set_xlabel('coverage')
axs[0].set_ylabel('fraction of bases sampled')
# plot cumulative coverage

axs[1].set_xlabel('coverage')
axs[1].set_ylabel('fraction of bases sampled >= coverage')
```





Computing the FRiP score

The FRiP score is defined as the fraction of reads that fall into a peak and is often used as a measure of ChIP-seq quality. For this example, we need a BED file containing the peak regions. Such files are usually computed using a peak caller. Also, two bam files are going to be used, corresponding to two biological replicates.

```
bed_files = ["peaks.bed"]
cr = countReadsPerBin.CountReadsPerBin([bam_file1, bam_file2],
                                        bedFile=bed_files,
                                        numberOfProcessors=10)
reads_at_peaks = cr.run()
print reads_at_peaks
```

```
array([[ 322.,  248.],
       [ 231.,  182.],
       [ 112.,  422.],
       ...,
       [ 120.,   76.],
       [ 235.,  341.],
       [ 246.,  265.]])
```

The result is a numpy array with a row for each peak region and a column for each BAM file.

```
reads_at_peaks.shape
```

```
(6295, 2)
```

Now, the total number of reads per peaks per bam file is computed:

```
total = reads_at_peaks.sum(axis=0)
```

Next, we need to find the total number of mapped reads in each of the bam files. For this we use the pysam module.

```
import pysam
bam1 = pysam.AlignmentFile(bam_file1)
bam2 = pysam.AlignmentFile(bam_file2)
```

Now, *bam1.mapped* and *bam2.mapped* contain the total number of mapped reads in each of the bam files, respectively.

Finally, we can compute the FRiP score:

```
frip1 = float(total[0]) / bam1.mapped
frip2 = float(total[1]) / bam2.mapped
print frip1, frip2
```

```
0.170030741997, 0.216740390353
```

Using mapReduce to sample paired-end fragment lengths

deepTools internally uses a map-reduce strategy, in which a computation is split into smaller parts that are sent to different processors. The output from the different processors is subsequently collated. The following example is based on the code available for *bamPEFragmentSize.py*

Here, we retrieve the reads from a BAM file and collect the fragment length. Reads are retrieved using pysam, and the *read* object returned contains the *template_length* attribute, which is the number of bases from the leftmost to the rightmost mapped base in the read pair.

First, we will create a function that can collect fragment lengths over a genomic position from a BAM file. As we will later call this function using mapReduce, the function accepts only one argument, namely a tuple with the parameters: chromosome name, start position, end position, and BAM file name.

```
import pysam
import numpy as np
def get_fragment_length(args):
    chrom, start, end, bam_file_name = args
    bam = pysam.AlignmentFile(bam_file_name)
    f_lens_list = []
    for fetch_start in range(start, end, 1000000):
        # simply get the reads over a region of 10000 bases
        fetch_end = min(end, fetch_start + 10000)

        f_lens_list.append(np.array([abs(read.template_length)
                                     for read in bam.fetch(chrom, fetch_start, fetch_end)
                                     if read.is_proper_pair and read.is_read1]))

    # concatenate all results
    return np.concatenate(f_lens_list)
```

Now, we can use *mapReduce* to call this function and compute fragment lengths over the whole genome. mapReduce needs to know the chromosome sizes, which can be easily retrieved from the BAM file. Furthermore, it needs to know the size of the region(s) sent to each processor. For this example, a region of 10 million bases is sent to each processor using the *genomeChunkLength* parameter. In other words, each processor executes the same *get_fragment_length* function to collect data over different 10 million base regions. The arguments to mapReduce are the list of arguments sent to the function, besides the first obligatory three (chrom start, end). In this case only one extra argument is passed to the function, the BAM file name. The next two positional arguments are the name of the function to call (*get_fragment_length*) and the chromosome sizes.

```
import deeptools.mapReduce
bam = pysam.AlignmentFile(bamFile)
chroms_sizes = list(zip(bam.references, bam.lengths))

result = mapReduce.mapReduce([bam_file_name],
                             get_fragment_length,
                             chroms_sizes,
                             genomeChunkLength=10000000,
                             numberOfProcessors=20,
                             verbose=True)

fragment_lengths =  np.concatenate(result)

print("mean fragment length {}".format(fragment_lengths.mean()))
print("median fragment length {}".format(np.median(fragment_lengths)))
```

```
0.170030741997, 0.216740390353
```

Indices and tables

- genindex
- modindex
- search

deepTools Galaxy. | code @ github.

1.10.2 deeptools package modules

deeptools.SES_scaleFactor module

class `deeptools.SES_scaleFactor.`**`Tester`**

Bases: `object`

`deeptools.SES_scaleFactor.`**`estimateScaleFactor`**(*bamFilesList, binLength, numberOfSamples, normalizationLength, avg_method='median', blackListFileName=None, numberOfProcessors=1, verbose=False, chrsToSkip=[], mappingStatsList=[]*)

Subdivides the genome into chunks to be analyzed in parallel using several processors. The code handles the creation of workers that compute fragment counts (coverage) for different regions and then collect and integrates the results.

bamFilesList [list] list of bam files to normalize

binLength [int] the window size in bp, where reads are going to be counted.

numberOfSamples [int] number of sites to sample from the genome. For more info see the documentation of the CountReadsPerBin class

normalizationLength [int] length, in bp, to normalize the data. For a value of 1, on average 1 read per base pair is found

avg_method [str] defines how the different values are to be summarized. The options are 'mean' and 'median'

chrsToSkip [list] name of the chromosomes to be excluded from the scale estimation. Usually the chrX is included.

blackListFileName [str] BED file containing blacklisted regions

mappingStatsList [list] List of the number of mapped reads per file

dict

Dictionary with the following keys:: 'size_factors' 'size_factors_based_on_mapped_reads' 'size_factors_SES' 'size_factors_based_on_mean' 'size_factors_based_on_median' 'mean' 'meanSES' 'median' 'reads_per_bin' 'std' 'sites_sampled'

```
>>> test = Tester()
>>> bin_length = 50
>>> num_samples = 4
>>> _dict = estimateScaleFactor([test.bamFile1, test.bamFile2], bin_length, num_
↪samples,  1)
>>> _dict['size_factors']
array([1. , 0.5])
>>> _dict['size_factors_based_on_mean']
array([1. , 0.5])
```

deeptools.bamHandler module

`deeptools.bamHandler.`**`countReadsInInterval`**(*args*)

`deeptools.bamHandler.`**`getMappingStats`**(*bam, nThreads*)

This is used for CRAM files, since idxstats() and .mapped/.unmapped are meaningless

This requires pysam > 0.13.0

`deeptools.bamHandler.`**`openBam`**(*bamFile*, *returnStats=False*, *nThreads=1*, *minimalDecoding=True*)

A wrapper for opening BAM/CRAM files.

bamFile: str A BAM/CRAM file name

returnStats: bool Return a tuple of (file_handle, nMappedReads, nUnmappedReads, statsDict). These additional values are needed by some downstream functions, since one can't use file_handle.mapped on CRAM files (or idxstats())

nThreads: int If returnStats is True, number of threads to use for computing statistics

minimalDecoding: Bool For CRAM files, don't decode the read name, sequence, qual, or auxiliary tag fields (these aren't used by most functions).

Returns either the file handle or a tuple as described in returnStats

deeptools.correlation module

`class` `deeptools.correlation.`**`Correlation`**(*matrix_file*, *corr_method=None*, *labels=None*, *remove_outliers=False*, *skip_zeros=False*, *log1p=False*)

Bases: `object`

class to work with matrices having sample data to compute correlations, plot them and make scatter plots

`compute_correlation`()

computes spearman or pearson correlation for the samples in the matrix

The matrix should contain the values of each sample per column that's why the transpose is used.

```
>>> matrix = np.array([[1, 2, 3, np.nan],
...                    [1, 2, 3, 4],
...                    [6, 4, 3, 1]]).T
>>> np.savez_compressed("/tmp/test_matrix.npz", matrix=matrix, labels=['a', 'b
↪', 'c'])
```

```
>>> c = Correlation("/tmp/test_matrix.npz", corr_method='pearson')
```

the results should be as in R

```
>>> c.compute_correlation().filled(np.nan)
array([[ 1.        ,  1.        , -0.98198051],
       [ 1.        ,  1.        , -0.98198051],
       [-0.98198051, -0.98198051,  1.        ]])
>>> c.corr_method = 'spearman'
>>> c.corr_matrix = None
>>> c.compute_correlation()
array([[ 1.,  1., -1.],
       [ 1.,  1., -1.],
       [-1., -1.,  1.]])
```

`static get_outlier_indices`(*data*, *max_deviation=200*)

The method is based on the median absolute deviation. See Boris Iglewicz and David Hoaglin (1993), "Volume 16: How to Detect and Handle Outliers", The ASQC Basic References in Quality Control: Statistical Techniques, Edward F. Mykytka, Ph.D., Editor.

returns the list, without the outliers

The max_deviation=200 is like selecting a z-score larger than 200, just that it is based on the median and the median absolute deviation instead of the mean and the standard deviation.

`load_matrix`(*matrix_file*)

loads a matrix file saved using the numpy savez method. Two keys are expected: 'matrix' and 'labels'. The matrix should contain one sample per row

`plot_correlation`(*plot_filename*, *plot_title=''*, *vmax=None*, *vmin=None*, *colormap='jet'*, *image_format=None*, *plot_numbers=False*, *plotWidth=11*, *plotHeight=9.5*)

plots a correlation using a symmetric heatmap

`plot_pca`(*plot_filename=None, PCs=[1, 2], plot_title='', image_format=None, log1p=False, plotWidth=5, plotHeight=10, cols=None, marks=None*)

Plot the PCA of a matrix

Returns the matrix of plotted values.

`plot_scatter`(*plot_filename*, *plot_title=''*, *image_format=None*, *log1p=False*, *xRange=None*, *yRange=None*)

Plot the scatter plots of a matrix in which each row is a sample

`plotly_correlation`(*corr_matrix*, *plot_filename*, *labels*, *plot_title=''*, *vmax=None*, *vmin=None*, *plot_numbers=True*, *colormap='jet'*)

plot_correlation, but using plotly

`plotly_pca`(*plotFile*, *Wt*, *pvar*, *PCs*, *eigenvalues*, *cols*, *plotTitle*)

A plotly version of plot_pca, that's called by it to do the actual plotting

`plotly_scatter`(*plot_filename*, *corr_matrix*, *plot_title=''*, *minXVal=None*, *maxXVal=None*, *minYVal=None*, *maxYVal=None*)

Make the scatter plot of a matrix with plotly

`remove_outliers`(*verbose=True*)

get the outliers *per column* using the median absolute deviation method

Returns the filtered matrix

`remove_rows_of_zeros`()

`save_corr_matrix`(*file_handle*)

saves the correlation matrix

deeptools.correlation_heatmap module

`deeptools.correlation_heatmap.`**`plot_correlation`**(*corr_matrix*, *labels*, *plotFileName*, *vmax=None*, *vmin=None*, *colormap='jet'*, *image_format=None*, *plot_numbers=False*, *plot_title=''*)

deeptools.countReadsPerBin module

class `deeptools.countReadsPerBin.`**`CountReadsPerBin`**(*bamFilesList*, *binLength=50*, *numberOfSamples=None*, *numberOfProcessors=1*, *verbose=False*, *region=None*, *bedFile=None*, *extendReads=False*, *genomeChunkSize=None*, *blackListFileName=None*, *minMappingQuality=None*, *ignoreDuplicates=False*, *chrsToSkip=[]*, *stepSize=None*, *center_read=False*, *samFlag_include=None*, *samFlag_exclude=None*, *zerosToNans=False*, *skipZeroOverZero=False*, *smoothLength=0*, *minFragmentLength=0*, *maxFragmentLength=0*, *out_file_for_raw_data=None*, *bed_and_bin=False*, *statsList=[]*, *mappedList=[]*)

Bases: `object`

Collects coverage over multiple bam files using multiprocessing

This function collects read counts (coverage) from several bam files and returns an numpy array with the results. This class uses multiprocessing to compute the coverage.

bamFilesList [list] List containing the names of indexed bam files. E.g. ['file1.bam', 'file2.bam']

binLength [int] Length of the window/bin. This value is overruled by `bedFile` if present.

numberOfSamples [int] Total number of samples. The genome is divided into `numberOfSamples`, each with a window/bin length equal to `binLength`. This value is overruled by `stepSize` in case such value is present and by `bedFile` in which case the number of samples and bins are defined in the bed file

numberOfProcessors [int] Number of processors to use. Default is 4

verbose [bool] Output messages. Default: False

region [str] Region to limit the computation in the form chrom:start:end.

bedFile [list of file_handles.] Each file handle corresponds to a bed file containing the regions for which to compute the coverage. This option overrules `binLength`, `numberOfSamples` and `stepSize`.

blackListFileName [str] A string containing a BED file with blacklist regions.

extendReads : bool, int

> Whether coverage should be computed for the extended read length (i.e. the region covered by the two mates or the regions expected to be covered by single-reads). If the value is 'int', then then this is interpreted as the fragment length to extend reads that are not paired. For Illumina reads, usual values are around 300. This value can be determined using the peak caller MACS2 or can be approximated by the fragment lengths computed when preparing the library for sequencing. If the value is of the variable is true and not value is given, the fragment size is sampled from the library but only if the library is paired-end. Default: False

minMappingQuality [int] Reads of a mapping quality less than the give value are not considered. Default: None

ignoreDuplicates [bool] Whether read duplicates (same start, end position. If paired-end, same start-end for mates) are to be excluded. Default: false

chrToSkip: list List with names of chromosomes that do not want to be included in the coverage computation. This is useful to remove unwanted chromosomes (e.g. 'random' or 'Het').

stepSize [int] the positions for which the coverage is computed are defined as follows: `range(start, end, stepSize)`. Thus, a stepSize of 1, will compute the coverage at each base pair. If the stepSize is equal to the binLength then the coverage is computed for consecutive bins. If seepSize is smaller than the binLength, then teh bins will overlap.

center_read [bool] Determines if reads should be centered with respect to the fragment length.

samFlag_include [int] Extracts only those reads having the SAM flag. For example, to get only reads that are the first mates a samFlag of 64 could be used. Similarly, the samFlag_include can be used to select only reads mapping on the reverse strand or to get only properly paired reads.

samFlag_exclude [int] Removes reads that match the SAM flag. For example to get all reads that map to the forward strand a samFlag_exlude 16 should be used. Which translates into exclude all reads that map to the reverse strand.

zerosToNans [bool] If true, zero values encountered are transformed to Nans. Default false.

skipZeroOverZero [bool] If true, skip bins where all input BAM files have no coverage (only applicable to bamCompare).

minFragmentLength [int] If greater than 0, fragments below this size are excluded.

maxFragmentLength [int] If greater than 0, fragments above this size are excluded.

out_file_for_raw_data [str] File name to save the raw counts computed

statsList [list] For each BAM file in bamFilesList, the associated per-chromosome statistics returned by openBam

mappedList [list] For each BAM file in bamFilesList, the number of mapped reads in the file.

bed_and_bin [boolean] If true AND a bedFile is given, compute coverage of each bin of the given size in each region of bedFile

genomeChunkSize [int] If not None, the length of the genome used for multiprocessing.

numpy array

Each row correspond to each bin/bed region and each column correspond to each of the bamFiles.

The test data contains reads for 200 bp.

```
>>> test = Tester()
```

The transpose function is used to get a nicer looking output. The first line corresponds to the number of reads per bin in bam file 1

```
>>> c = CountReadsPerBin([test.bamFile1, test.bamFile2], 50, 4)
>>> np.transpose(c.run())
array([[0., 0., 1., 1.],
       [0., 1., 1., 2.]])
```

`count_reads_in_region`(*chrom*, *start*, *end*, *bed_regions_list=None*)

Counts the reads in each bam file at each 'stepSize' position within the interval (start, end) for a window or bin of size binLength.

The stepSize controls the distance between bins. For example, a step size of 20 and a bin size of 20 will create bins next to each other. If the step size is smaller than the bin size the bins will overlap.

If a list of bedRegions is given, then the number of reads that overlaps with each region is counted.

chrom [str] Chrom name

start [int] start coordinate

end [int] end coordinate

bed_regions_list: list List of list of tuples of the form (start, end) corresponding to bed regions to be processed. If not bed file was passed to the object constructor then this list is empty.

numpy array The result is a numpy array that as rows each bin and as columns each bam file.

Initialize some useful values

```
>>> test = Tester()
>>> c = CountReadsPerBin([test.bamFile1, test.bamFile2], 25, 0, stepSize=50)
```

The transpose is used to get better looking numbers. The first line corresponds to the number of reads per bin in the first bamfile.

```
>>> _array, __ = c.count_reads_in_region(test.chrom, 0, 200)
>>> _array
array([[0., 0.],
       [0., 1.],
       [1., 1.],
       [1., 2.]])
```

`getReadLength`(*read*)

`getSmoothRange`(*tileIndex*, *tileSize*, *smoothRange*, *maxPosition*)

Given a tile index position and a tile size (length), return the a new indices over a larger range, called the smoothRange. This region is centered in the tileIndex an spans on both sizes to cover the smoothRange. The smoothRange is trimmed in case it is less than zero or greater than maxPosition

```
---------------|==================|------------------
           tileStart
      |--------------------------------------|
      |    <--      smoothRange     -->      |
      |
tileStart - (smoothRange-tileSize)/2
```

Test for a smooth range that spans 3 tiles.

```
>>> c = CountReadsPerBin([], 1, 1, 1, 0)
>>> c.getSmoothRange(5, 1, 3, 10)
(4, 7)
```

Test smooth range truncated on start.

```
>>> c.getSmoothRange(0, 10, 30, 200)
(0, 2)
```

Test smooth range truncated on start.

```
>>> c.getSmoothRange(1, 10, 30, 4)
(0, 3)
```

Test smooth range truncated on end.

```
>>> c.getSmoothRange(5, 1, 3, 5)
(4, 5)
```

Test smooth range not multiple of tileSize.

```
>>> c.getSmoothRange(5, 10, 24, 10)
(4, 6)
```

get_chunk_length(*bamFilesHandles*, *genomeSize*, *chromSizes*, *chrLengths*)

get_coverage_of_region(*bamHandle*, *chrom*, *regions*, *fragmentFromRead_func=None*)

Returns a numpy array that corresponds to the number of reads that overlap with each tile.

```
>>> test = Tester()
>>> import pysam
>>> c = CountReadsPerBin([], stepSize=1, extendReads=300)
```

For this case the reads are length 36. The number of overlapping read fragments is 4 and 5 for the positions tested.

```
>>> c.get_coverage_of_region(pysam.AlignmentFile(test.bamFile_PE), 'chr2',
... [(5000833, 5000834), (5000834, 5000835)])
array([4., 5.])
```

In the following example a paired read is extended to the fragment length which is 100 The first mate starts at 5000000 and the second at 5000064. Each mate is extended to the fragment length *independently* At position 500090-500100 one fragment of length 100 overlap, and after position 5000101 there should be zero reads.

```
>>> c.zerosToNans = True
>>> c.get_coverage_of_region(pysam.AlignmentFile(test.bamFile_PE), 'chr2',
... [(5000090, 5000100), (5000100, 5000110)])
array([ 1., nan])
```

In the following case the reads length is 50. Reads are not extended.

```
>>> c.extendReads=False
>>> c.get_coverage_of_region(pysam.AlignmentFile(test.bamFile2), '3R', [(148,␣
↪150), (150, 152), (152, 154)])
array([1., 2., 2.])
```

get_fragment_from_read(*read*)

Get read start and end position of a read. If given, the reads are extended as follows: If reads are paired end, each read mate is extended to match the fragment length, otherwise, a default fragment length is used. If reads are split (give by the CIGAR string) then the multiple positions of the read are returned. When reads are extended the cigar information is skipped.

read: pysam object.

The following values are defined (for forward reads):

```
     |--          -- read.tlen --              --|
     |-- read.alen --|
-----|===============>------------<==============|----
     |               |            |
read.reference_start
```

(continues on next page)

(continued from previous page)

```
            read.reference_end  read.pnext

  and for reverse reads


     |--               -- read.tlen --            --|
                                 |-- read.alen --|
-----|===============>-----------<===============|----
     |                           |               |
  read.pnext             read.reference_start  read.reference_end
```

this is a sketch of a pair-end reads

The function returns the fragment start and end, either using the paired end information (if available) or extending the read in the appropriate direction if this is single-end.

read : pysam read object

list of tuples [(fragment start, fragment end)]

```
>>> test = Tester()
>>> c = CountReadsPerBin([], 1, 1, 200, extendReads=True)
>>> c.defaultFragmentLength=100
>>> c.get_fragment_from_read(test.getRead("paired-forward"))
[(5000000, 5000100)]
>>> c.get_fragment_from_read(test.getRead("paired-reverse"))
[(5000000, 5000100)]
>>> c.defaultFragmentLength = 200
>>> c.get_fragment_from_read(test.getRead("single-forward"))
[(5001491, 5001691)]
>>> c.get_fragment_from_read(test.getRead("single-reverse"))
[(5001536, 5001736)]
>>> c.defaultFragmentLength = 'read length'
>>> c.get_fragment_from_read(test.getRead("single-forward"))
[(5001491, 5001527)]
>>> c.defaultFragmentLength = 'read length'
>>> c.extendReads = False
>>> c.get_fragment_from_read(test.getRead("paired-forward"))
[(5000000, 5000036)]
```

Tests for read centering.

```
>>> c = CountReadsPerBin([], 1, 1, 200, extendReads=True, center_read=True)
>>> c.defaultFragmentLength = 100
>>> assert(c.get_fragment_from_read(test.getRead("paired-forward")) ==␣
↪[(5000032, 5000068)])
>>> c.defaultFragmentLength = 200
>>> assert(c.get_fragment_from_read(test.getRead("single-reverse")) ==␣
↪[(5001618, 5001654)])
```

`static is_proper_pair`(*read*, *maxPairedFragmentLength*)

Checks if a read is proper pair meaning that both mates are facing each other and are in the same chromosome and are not to far away. The sam flag for proper pair can not always be trusted. Note that if the fragment size is > maxPairedFragmentLength (~2kb usually) that False will be returned. :return: bool

```
>>> import pysam
>>> import os
```

(continues on next page)

(continued from previous page)

```
>>> from deeptools.countReadsPerBin import CountReadsPerBin as cr
>>> root = os.path.dirname(os.path.abspath(__file__)) + "/test/test_data/"
>>> bam = pysam.AlignmentFile("{}/test_proper_pair_filtering.bam".
↪format(root))
>>> iter = bam.fetch()
>>> read = next(iter)
>>> cr.is_proper_pair(read, 1000) # "keep" read
True
>>> cr.is_proper_pair(read, 200) # "keep" read, but maxPairedFragmentLength␣
↪is too short
False
>>> read = next(iter)
>>> cr.is_proper_pair(read, 1000) # "improper pair"
False
>>> read = next(iter)
>>> cr.is_proper_pair(read, 1000) # "mismatch chr"
False
>>> read = next(iter)
>>> cr.is_proper_pair(read, 1000) # "same orientation1"
False
>>> read = next(iter)
>>> cr.is_proper_pair(read, 1000) # "same orientation2"
False
>>> read = next(iter)
>>> cr.is_proper_pair(read, 1000) # "rev first"
False
>>> read = next(iter)
>>> cr.is_proper_pair(read, 1000) # "rev first OK"
True
>>> read = next(iter)
>>> cr.is_proper_pair(read, 1000) # "for first"
False
>>> read = next(iter)
>>> cr.is_proper_pair(read, 1000) # "for first"
True
```

run(*allArgs=None*)

class deeptools.countReadsPerBin.**Tester**

Bases: `object`

getRead(*readType*)

prepare arguments for test

deeptools.countReadsPerBin.**countReadsInRegions_wrapper**(*args*)

Passes the arguments to countReadsInRegions_worker. This is a step required given the constrains from the multiprocessing module. The args var, contains as first element the 'self' value from the countReadsPerBin object

deeptools.countReadsPerBin.**estimateSizeFactors**(*m*)

Compute size factors in the same way as DESeq2. The inverse of that is returned, as it's then compatible with bamCoverage.

m : a numpy ndarray

```
>>> m = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [0, 10, 0], [10, 5, 100]])
>>> sf = estimateSizeFactors(m)
>>> assert(np.all(np.abs(sf - [1.305, 0.9932, 0.783]) < 1e-4))
```

(continues on next page)

(continued from previous page)

```
>>> m = np.array([[0, 0], [0, 1], [1, 1], [1, 2]])
>>> sf = estimateSizeFactors(m)
>>> assert(np.all(np.abs(sf - [1.1892, 0.8409]) < 1e-4))
```

`deeptools.countReadsPerBin.`**`remove_row_of_zeros`**`(`*matrix*`)`

deeptools.deepBlue

class `deeptools.deepBlue.`**`deepBlue`**`(`*sample*, *url='http://deepblue.mpi-inf.mpg.de/xmlrpc'*, *userKey='anonymous_key'*`)`

Bases: `object`

`chroms``(`*chrom=None*`)`

Like the chroms() function in pyBigWig, returns either chromsDict (chrom is None) or the length of a given chromosome

`close``()`

`getChroms``()`

Determines and sets the chromosome names/sizes for a given sample. On error, this raises a runtime exception.

self.chroms is then a dictionary of chromosome:length pairs

`getEID``()`

Given a sample name, return its associated experiment ID (or None on error).

self.experimentID is then the internal ID (e.g., e52525)

`getGenome``()`

Determines and sets the genome assigned to a given sample. On error, this raises a runtime exception.

self.genome is then the internal genome ID.

`preload``(`*regions*, *tmpDir=None*`)`

Given a sample and a set of regions, write a bigWig file containing the underlying signal.

This function returns the file name, which needs to be deleted by the calling function at some point.

This sends queries one chromosome at a time, due to memory limits on deepBlue

`deeptools.deepBlue.`**`isDeepBlue`**`(`*fname*`)`

Returns true if the file ends in .wig, .wiggle, or .bedgraph, since these indicate a file on the deepBlue server

`deeptools.deepBlue.`**`makeChromTiles`**`(`*db*`)`

Make a region for each chromosome

`deeptools.deepBlue.`**`makeRegions`**`(`*BED*, *args*`)`

Given a list of BED/GTF files, make a list of regions. These are vaguely extended as appropriate. For simplicity, the maximum of –beforeRegionStartLength and –afterRegionStartLength are tacked on to each end and transcripts are used for GTF files.

`deeptools.deepBlue.`**`makeTiles`**`(`*db*, *args*`)`

Given a deepBlue object, return a list of regions that will be queried

`deeptools.deepBlue.`**`mergeRegions`**`(`*regions*`)`

Given a list of [(chrom, start, end), . . .], merge all overlapping regions

This returns a dict, where values are sorted lists of [start, end].

`deeptools.deepBlue.`**`preloadWrapper`**(*foo*)

This is a wrapper around the preload function for multiprocessing

deeptools.getFragmentAndReadSize module

`deeptools.getFragmentAndReadSize.`**`getFragmentLength_worker`**(*chrom*, *start*, *end*, *bamFile*, *distanceBetweenBins*)

Queries the reads at the given region for the distance between reads and the read length

chrom [str] chromosome name

start [int] region start

end [int] region end

bamFile [str] BAM file name

distanceBetweenBins [int] the number of bases at the end of each bin to ignore

np.array an np.array, where first column is fragment length, the second is for read length

`deeptools.getFragmentAndReadSize.`**`getFragmentLength_wrapper`**(*args*)

`deeptools.getFragmentAndReadSize.`**`get_read_and_fragment_length`**(*bamFile*, *return_lengths=False*, *blackListFileName=None*, *binSize=50000*, *distanceBetweenBins=1000000*, *numberOfProcessors=None*, *verbose=False*)

Estimates the fragment length and read length through sampling

bamFile [str] BAM file name

return_lengths : bool numberOfProcessors : int verbose : bool binSize : int distanceBetweenBins : int

d [dict] tuple of two dictionaries, one for the fragment length and the other for the read length. The dictionaries summarise the mean, median etc. values

deeptools.getRatio module

`deeptools.getRatio.`**`compute_ratio`**(*value1*, *value2*, *args*)

`deeptools.getRatio.`**`getRatio`**(*tileCoverage*, *args*)

The mapreduce method calls this function for each tile. The parameters (args) are fixed in the main method.

```
>>> funcArgs= {'valueType': 'ratio', 'scaleFactors': (1,1), 'pseudocount': [1, 1]}
>>> getRatio([9, 19], funcArgs)
0.5
>>> getRatio([0, 0], funcArgs)
1.0
>>> getRatio([np.nan, np.nan], funcArgs)
nan
>>> getRatio([np.nan, 1.0], funcArgs)
```

(continues on next page)

(continued from previous page)

```
nan
>>> funcArgs['valueType'] ='subtract'
>>> getRatio([20, 10], funcArgs)
10
>>> funcArgs['scaleFactors'] = (1, 0.5)
>>> getRatio([10, 20], funcArgs)
0.0
```

The reciprocal ratio is of a and b is: is a/b if a/b > 1 else -1* b/a >>> funcArgs['valueType'] ='reciprocal_ratio' >>> funcArgs['scaleFactors'] = (1, 1) >>> funcArgs['pseudocount'] = [0, 0] >>> getRatio([2, 1], funcArgs) 2.0 >>> getRatio([1, 2], funcArgs) -2.0 >>> getRatio([1, 1], funcArgs) 1.0

deeptools.getScorePerBigWigBin module

class `deeptools.getScorePerBigWigBin.`**`Tester`**

Bases: `object`

`deeptools.getScorePerBigWigBin.`**`countFragmentsInRegions_worker`**(*chrom*, *start*, *end*, *bigWigFiles*, *stepSize*, *binLength*, *save_data*, *bedRegions=None*)

returns the average score in each bigwig file at each 'stepSize' position within the interval start, end for a 'binLength' window. Because the idea is to get counts for window positions at different positions for sampling the bins are equally spaced and *not adjacent*.

If a list of bedRegions is given, then the number of reads that overlaps with each region is counted.

Test dataset with two samples covering 200 bp. >>> test = Tester()

Fragment coverage. >>> np.transpose(countFragmentsInRegions_worker(test.chrom, 0, 200, [test.bwFile1, test.bwFile2], 50, 25, False)[0]) array([[1., 1., 2., 2.],

> [1., 1., 1., 3.]])

```
>>> np.transpose(countFragmentsInRegions_worker(test.chrom, 0, 200, [test.bwFile1,
↪ test.bwFile2], 200, 200, False)[0])
array([[1.5],
       [1.5]])
```

BED regions: >>> bedRegions = [[test.chrom, [(45, 55)]], [test.chrom, [(95, 105)]], [test.chrom, [(145, 155)]]] >>> np.transpose(countFragmentsInRegions_worker(test.chrom, 0, 200,[test.bwFile1, test.bwFile2], 200, 200, False, … bedRegions=bedRegions)[0]) array([[1. , 1.5, 2.],

> [1. , 1. , 2.]])

`deeptools.getScorePerBigWigBin.`**`countReadsInRegions_wrapper`**(*args*)

`deeptools.getScorePerBigWigBin.`**`getChromSizes`**(*bigwigFilesList*)

Get chromosome sizes from bigWig file with pyBigWig

Test dataset with two samples covering 200 bp. >>> test = Tester()

Chromosome name(s) and size(s). >>> assert(getChromSizes([test.bwFile1, test.bwFile2]) == ([('3R', 200)], set([])))

`deeptools.getScorePerBigWigBin.`**`getScorePerBin`**(*bigWigFiles*, *binLength*, *numberOfProcessors=1*, *verbose=False*, *region=None*, *bedFile=None*, *blackListFileName=None*, *stepSize=None*, *chrsToSkip=[]*, *out_file_for_raw_data=None*, *allArgs=None*)

This function returns a matrix containing scores (median) for the coverage of fragments within a region. Each row corresponds to a sampled region. Likewise, each column corresponds to a bigwig file.

Test dataset with two samples covering 200 bp. >>> test = Tester() >>> np.transpose(getScorePerBin([test.bwFile1, test.bwFile2], 50, 3)) array([[1., 1., 2., 2.],

> [1., 1., 1., 3.]])

deeptools.heatmapper module

`deeptools.heatmapper.`**`chopRegions`**(*exonsInput*, *left=0*, *right=0*)

exons is a list of (start, end) tuples. The goal is to chop these into separate lists of tuples, to take care or unscaled regions. "left" and "right" denote regions of a given size to exclude from the normal binning process (unscaled regions).

This outputs three lists of (start, end) tuples:

leftBins: 5' unscaled regions bodyBins: body bins for scaling rightBins: 3' unscaled regions

In addition are two integers padLeft: Number of bases of padding on the left (due to not being able to fulfill "left") padRight: As above, but on the right side

`deeptools.heatmapper.`**`chopRegionsFromMiddle`**(*exonsInput*, *left=0*, *right=0*)

Like chopRegions(), above, but returns two lists of tuples on each side of the center point of the exons.

The steps are as follow:

1) Find the center point of the set of exons (e.g., [(0, 200), (300, 400), (800, 900)] would be centered at 200)

   - If a given exon spans the center point then the exon is split

2) The given number of bases at the end of the left-of-center list are extracted

   - If the set of exons don't contain enough bases, then padLeft is incremented accordingly

3) As above but for the right-of-center list
4) A tuple of (#2, #3, pading on the left, and padding on the right) is returned

`deeptools.heatmapper.`**`computeSilhouetteScore`**(*d*, *idx*, *labels*)

Given a square distance matrix with NaN diagonals, compute the silhouette score of a given row (idx). Each row should have an associated label (labels).

`deeptools.heatmapper.`**`compute_sub_matrix_wrapper`**(*args*)

`class` `deeptools.heatmapper.`**`heatmapper`**

Bases: `object`

Class to handle the reading and plotting of matrices.

`static change_chrom_names`(*chrom*)

Changes UCSC chromosome names to ensembl chromosome names and vice versa.

computeMatrix(*score_file_list*, *regions_file*, *parameters*, *blackListFileName=None*, *verbose=False*, *allArgs=None*)

Splits into multiple cores the computation of the scores per bin for each region (defined by a hash '#' in the regions (BED/GFF) file.

static **compute_sub_matrix_worker**(*self*, *chrom*, *start*, *end*, *score_file_list*, *parameters*, *regions*)

numpy matrix A numpy matrix that contains per each row the values found per each of the regions given

static **coverage_from_array**(*valuesArray*, *zones*, *binSize*, *avgType*)

static **coverage_from_big_wig**(*bigwig*, *chrom*, *zones*, *binSize*, *avgType*, *nansAsZeros=False*, *verbose=True*)

uses pyBigWig to query a region define by chrom and zones. The output is an array that contains the bigwig value per base pair. The summary over bins is done in a later step when coverage_from_array is called. This method is more reliable than querying the bins directly from the bigwig, which should be more efficient.

By default, any region, even if no chromosome match is found on the bigwig file, produces a result. In other words no regions are skipped.

zones: array as follows zone0: region before the region start,

> zone1: 5' unscaled region (if present) zone2: the body of the region (not always present) zone3: 3' unscaled region (if present) zone4: the region from the end of the region downstream

each zone is a tuple containing start, end, and number of bins

This is useful if several matrices wants to be merged or if the sorted BED output of one computeMatrix operation needs to be used for other cases

getTicks(*idx*)

This is essentially a wrapper around getProfileTicks to accomdate the fact that each column has its own ticks.

get_individual_matrices(*matrix*)

In case multiple matrices are saved one after the other this method splits them appart. Returns a list containing the matrices

get_num_individual_matrix_cols()

returns the number of columns that each matrix should have. This is done because the final matrix that is plotted can be composed of smaller matrices that are merged one after the other.

static **matrix_avg**(*matrix*, *avgType='mean'*)

matrix_from_dict(*matrixDict*, *regionsDict*, *parameters*)

static **my_average**(*valuesArray*, *avgType='mean'*)

computes the mean, median, etc but only for those values that are not Nan

read_matrix_file(*matrix_file*)

save_BED(*file_handle*)

save_matrix(*file_name*)

saves the data required to reconstruct the matrix the format is: A header containing the parameters used to create the matrix encoded as: @key:value key2:value2 etc... The rest of the file has the same first 5 columns of a BED file: chromosome name, start, end, name, score and strand, all separated by tabs. After the fifth column the matrix values are appended separated by tabs. Groups are separated by adding a line starting with a hash (#) and followed by the group name.

The file is gzipped.

save_matrix_values (*file_name*)

save_tabulated_values (*file_handle*, *reference_point_label='TSS'*, *start_label='TSS'*, *end_label='TES'*, *averagetype='mean'*)

Saves the values averaged by col using the avg_type given

Args: file_handle: file name to save the file reference_point_label: Name of the reference point label start_label: Name of the star label end_label: Name of the end label averagetype: average type (e.g. mean, median, std)

deeptools.heatmapper.**trimZones** (*zones*, *maxLength*, *binSize*, *padRight*)

Given a (variable length) list of lists of (start, end) tuples, trim/remove and tuple that extends past maxLength (e.g., the end of a chromosome)

Returns the trimmed zones and padding

deeptools.heatmapper_utilities module

deeptools.heatmapper_utilities.**getProfileTicks** (*hm*, *referencePointLabel*, *startLabel*, *endLabel*, *idx*)

returns the position and labelling of the xticks that correspond to the heatmap

As of deepTools 3, the various parameters can be lists, in which case we then need to index things (the idx parameter)

As of matplotlib 3 the ticks in the heatmap need to have 0.5 added to them.

As of matplotlib 3.1 there is no longer padding added to all ticks. Reference point ticks will be adjusted by width/2 or width for spacing and the last half of scaled ticks will be shifed by 1 bin so the ticks are at the beginning of bins.

deeptools.heatmapper_utilities.**plot_single** (*ax*, *ma*, *average_type*, *color*, *label*, *plot_type='lines'*)

Adds a line to the plot in the given ax using the specified method

ax [matplotlib axis] matplotlib axis

ma [numpy array] numpy array The data on this matrix is summarized according to the *average_type* argument.

average_type [str] string values are sum mean median min max std

color [str] a valid color: either a html color name, hex (e.g #002233), RGB + alpha tuple or list or RGB tuple or list

label [str] label

plot_type: str type of plot. Either 'se' for standard error, 'std' for standard deviation, 'overlapped_lines' to plot each line of the matrix, fill to plot the area between the x axis and the value or any other string to just plot the average line.

ax matplotlib axis

```
>>> import matplotlib.pyplot as plt
>>> import os
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
>>> matrix = np.array([[1,2,3],
...                    [4,5,6],
...                    [7,8,9]])
>>> ax = plot_single(ax, matrix -2, 'mean', color=[0.6, 0.8, 0.9], label='fill␣
↪light blue', plot_type='fill')
```

(continues on next page)

(continued from previous page)

```
>>> ax = plot_single(ax, matrix, 'mean', color='blue', label='red')
>>> ax = plot_single(ax, matrix + 5, 'mean', color='red', label='red', plot_type=
↪'std')
>>> ax = plot_single(ax, matrix + 10, 'mean', color='#cccccc', label='gray se',␣
↪plot_type='se')
>>> ax = plot_single(ax, matrix + 20, 'mean', color=(0.9, 0.5, 0.9), label='violet
↪', plot_type='std')
>>> ax = plot_single(ax, matrix + 30, 'mean', color=(0.9, 0.5, 0.9, 0.5), label=
↪'violet with alpha', plot_type='std')
>>> leg = ax.legend()
>>> plt.savefig("/tmp/test.pdf")
>>> plt.close()
>>> fig = plt.figure()
>>> os.remove("/tmp/test.pdf")
```

`deeptools.heatmapper_utilities.`**`plotly_single`**(*ma*, *average_type*, *color*, *label*, *plot_type='line'*)

A plotly version of plot_single. Returns a list of traces

deeptools.mapReduce module

`deeptools.mapReduce.`**`blSubtract`**(*t*, *chrom*, *chunk*)

If a genomic region overlaps with a blacklisted region, then subtract that region out

returns a list of lists

`deeptools.mapReduce.`**`getUserRegion`**(*chrom_sizes*, *region_string*, *max_chunk_size=1000000.0*)

Verifies if a given region argument, given by the user is valid. The format of the region_string is chrom:start:end:tileSize where start, end and tileSize are optional.

Parameters

- **`chrom_sizes`** – dictionary of chromosome/scaffold size. Key=chromosome name
- **`region_string`** – a string of the form chr:start:end
- **`max_chunk_size`** – upper limit for the chunk size

Returns tuple chrom_size for the region start, region end, chunk size

#>>> data = getUserRegion({'chr2': 1000}, "chr1:10:10") #Traceback (most recent call last): # ... #NameError: Unknown chromosome: chr1 #Known chromosomes are: ['chr2']

If the region end is biger than the chromosome size, this value is used instead >>> getUserRegion({'chr2': 1000}, "chr2:10:1001") ([('chr2', 1000)], 10, 1000, 990)

Test chunk and regions size reduction to match tile size >>> getUserRegion({'chr2': 200000}, "chr2:10:123344:3") ([('chr2', 123344)], 9, 123345, 123336)

Test chromosome name mismatch >>> getUserRegion({'2': 200000}, "chr2:10:123344:3") ([('2', 123344)], 9, 123345, 123336) >>> getUserRegion({'chrM': 200000}, "MT:10:123344:3") ([('chrM', 123344)], 9, 123345, 123336)

`deeptools.mapReduce.`**`mapReduce`**(*staticArgs*, *func*, *chromSize*, *genomeChunkLength=None*, *region=None*, *bedFile=None*, *blackListFileName=None*, *numberOfProcessors=4*, *verbose=False*, *includeLabels=False*, *keepExons=False*, *transcriptID='transcriptID'*, *exonID='exonID'*, *transcript_id_designator='transcript_id'*, *self_=None*)

Split the genome into parts that are sent to workers using a defined number of procesors. Results are collected and returned.

For each genomic region the given 'func' is called using the following parameters:

chrom, start, end, staticArgs

The *arg* are static, *pickable* variables that need to be sent to workers.

The genome chunk length corresponds to a fraction of the genome, in bp, that is send to each of the workers for processing.

Depending on the type of process a larger or shorter regions may be preferred

Parameters

- **`chromSize`** – A list of duples containing the chromosome name and its length
- **`region`** – The format is chr:start:end:tileSize (see function getUserRegion)
- **`staticArgs`** – tuple of arguments that are sent to the given 'func'
- **`func`** – function to call. The function is called using the following parameters (chrom, start, end, staticArgs)
- **`bedFile`** – Is a bed file is given, the args to the func to be called are extended to include a list of bed defined regions.
- **`blackListFileName`** – A list of regions to exclude from all computations. Note that this has genomeChunkLength resolution…
- **`self`** – In case mapreduce should make a call to an object the self variable has to be passed.
- **`includeLabels`** – Pass group and transcript labels into the calling function. These are added to the static args (groupLabel and transcriptName).

If "includeLabels" is true, a tuple of (results, labels) is returned

deeptools.sumCoveragePerBin

`class` `deeptools.sumCoveragePerBin.`**`SumCoveragePerBin`**(*bamFilesList, binLength=50, numberOfSamples=None, numberOfProcessors=1, verbose=False, region=None, bedFile=None, extendReads=False, genomeChunkSize=None, blackListFileName=None, minMappingQuality=None, ignoreDuplicates=False, chrsToSkip=[], stepSize=None, center_read=False, samFlag_include=None, samFlag_exclude=None, zerosToNans=False, skipZeroOverZero=False, smoothLength=0, minFragmentLength=0, maxFragmentLength=0, out_file_for_raw_data=None, bed_and_bin=False, statsList=[], mappedList=[]*)

Bases: `deeptools.countReadsPerBin.CountReadsPerBin`

This is an extension of CountReadsPerBin for use with plotFingerprint. There, we need to sum the per-base coverage.

get_coverage_of_region(*bamHandle*, *chrom*, *regions*, *fragmentFromRead_func=None*)
Returns a numpy array that corresponds to the number of reads that overlap with each tile.

```
>>> test = Tester()
>>> import pysam
>>> c = SumCoveragePerBin([], stepSize=1, extendReads=300)
```

For this case the reads are length 36. The number of overlapping read fragments is 4 and 5 for the positions tested. Note that reads are NOT extended, due to there being a 0 length input list of BAM files!

```
>>> c.get_coverage_of_region(pysam.AlignmentFile(test.bamFile_PE), 'chr2',
... [(5000833, 5000834), (5000834, 5000835)])
array([4., 5.])
```

In the following case the reads length is 50. Reads are not extended.

```
>>> c.extendReads=False
>>> c.get_coverage_of_region(pysam.AlignmentFile(test.bamFile2), '3R', [(148,␣
↪150), (150, 152), (152, 154)])
array([2., 4., 4.])
```

class deeptools.sumCoveragePerBin.**Tester**
Bases: `object`

deeptools.utilities module

deeptools.utilities.**bam_blacklisted_reads**(*bam_handle*, *chroms_to_ignore*, *blackListFileName=None*, *numberOfProcessors=1*)

deeptools.utilities.**bam_blacklisted_worker**(*args*)

deeptools.utilities.**bam_total_reads**(*bam_handle*, *chroms_to_ignore*, *stats*)
Count the total number of mapped reads in a BAM file, filtering the chromosome given in chroms_to_ignore list

deeptools.utilities.**convertCmap**(*c*, *vmin=0*, *vmax=1*)

deeptools.utilities.**copyFileInMemory**(*filePath*, *suffix=”*)
copies a file into the special /dev/shm device which moves the file into memory. This process speeds ups the multiprocessor access to such files

deeptools.utilities.**getCommonChrNames**(*bamFileHandles*, *verbose=True*)
Compares the names and lengths of a list of bam file handles. The input is list of pysam file handles.

The function returns a duple containing the common chromosome names and the common chromome lengths.

Hopefully, only _random and chrM are not common.

deeptools.utilities.**getGC_content**(*tb*, *chrom*, *fragStart*, *fragEnd*, *fraction=True*)

deeptools.utilities.**getTLen**(*read*, *notAbs=False*)
Get the observed template length of a read. For a paired-end read, this is normally just the TLEN field. For SE reads this is the observed coverage of the genome (excluding splicing).

deeptools.utilities.**getTempFileName**(*suffix=”*)
Return a temporary file name. The calling function is responsible for deleting this upon completion.

deeptools.utilities.**gtfOptions**(*allArgs=None*)
This is used a couple places to setup arguments to mapReduce

`deeptools.utilities.`**`mungeChromosome`**(*chrom*, *chromList*)

A generic chromosome munging function. “chrom” is munged by adding/removing “chr” such that it appears in chromList

On error, None is returned, but a common chromosome list should be used beforehand to avoid this possibility

`deeptools.utilities.`**`smartLabel`**(*label*)

Given a file name, likely with a path, return the file name without the path and with the file extension removed. Thus, something like /path/to/some.special.file should return some.special, since only the first extension (if present) should be stripped.

`deeptools.utilities.`**`smartLabels`**(*labels*)

`deeptools.utilities.`**`tbitToBamChrName`**(*tbitNames*, *bamNames*)

checks if the chromosome names from the two-bit and bam file coincide. In case they do not coincide, a fix is tried. If successful, then a mapping table is returned. tbitNames and bamNames should be lists

`deeptools.utilities.`**`toBytes`**(*s*)

Like toString, but for functions requiring bytes in python3

`deeptools.utilities.`**`toString`**(*s*)

This takes care of python2/3 differences

deeptools.writeBedGraph module

`class` `deeptools.writeBedGraph.`**`WriteBedGraph`**(*bamFilesList*, *binLength=50*, *numberOfSamples=None*, *numberOfProcessors=1*, *verbose=False*, *region=None*, *bedFile=None*, *extendReads=False*, *genomeChunkSize=None*, *blackListFileName=None*, *minMappingQuality=None*, *ignoreDuplicates=False*, *chrsToSkip=[]*, *stepSize=None*, *center_read=False*, *samFlag_include=None*, *samFlag_exclude=None*, *zerosToNans=False*, *skipZeroOverZero=False*, *smoothLength=0*, *minFragmentLength=0*, *maxFragmentLength=0*, *out_file_for_raw_data=None*, *bed_and_bin=False*, *statsList=[]*, *mappedList=[]*)

Bases: `deeptools.countReadsPerBin.CountReadsPerBin`

Reads bam files coverages and writes a bedgraph or bigwig file

Extends the CountReadsPerBin object such that the coverage of bam files is writen to multiple bedgraph files at once.

The bedgraph files are later merge into one and converted into a bigwig file if necessary.

The constructor arguments are the same as for CountReadsPerBin. However, when calling the *run* method, the following parameters have to be passed

Given the following distribution of reads that cover 200 on a chromosome named ‘3R’:

```
  0                             100                           200
  |------------------------------------------------------------|
A                               ===============
                                               ===============

```

(continues on next page)

(continued from previous page)

```
B                 ===============               ===============
                                 ===============
                                                ===============
```

```
>>> import tempfile
>>> test_path = os.path.dirname(os.path.abspath(__file__)) + "/test/test_data/"
```

```
>>> outFile = tempfile.NamedTemporaryFile()
>>> bam_file = test_path +  "testA.bam"
```

For the example a simple scaling function is going to be used. This function takes the coverage found at each region and multiplies it to the scaling factor. In this case the scaling factor is 1.5

```
>>> function_to_call = scaleCoverage
>>> funcArgs = {'scaleFactor': 1.5}
```

Restrict process to a region between positions 0 and 200 of chromosome 3R

```
>>> region = '3R:0:200'
```

Set up such that coverage is computed for consecutive bins of length 25 bp >>> bin_length = 25 >>> step_size = 25

```
>>> num_sample_sites = 0 #overruled by step_size
>>> c = WriteBedGraph([bam_file], binLength=bin_length, region=region,␣
↪stepSize=step_size)
>>> c.run(function_to_call, funcArgs, outFile.name)
>>> f = open(outFile.name, 'r')
>>> f.readlines()
['3R\t0\t100\t0\n', '3R\t100\t200\t1.5\n']
>>> f.close()
>>> outFile.close()
```

run (*func_to_call*, *func_args*, *out_file_name*, *blackListFileName=None*, *format='bedgraph'*, *smoothLength=0*)

Given a list of bamfiles, a function and a function arguments, this method writes a bedgraph file (or bigwig) file for a partition of the genome into tiles of given size and a value for each tile that corresponds to the given function and that is related to the coverage underlying the tile.

func_to_call [str] function name to be called to convert the list of coverages computed for each bam file at each position into a single value. An example is a function that takes the ratio between the coverage of two bam files.

func_args [dict] dict of arguments to pass to *func*. E.g. {'scaleFactor':1.0}

out_file_name [str] name of the file to save the resulting data.

smoothLength [int] Distance in bp for smoothing the coverage per tile.

writeBedGraph_worker (*chrom*, *start*, *end*, *func_to_call*, *func_args*, *bed_regions_list=None*)

Writes a bedgraph based on the read coverage found on bamFiles

The given func is called to compute the desired bedgraph value using the funcArgs

chrom [str] Chrom name

start [int] start coordinate

end [int] end coordinate

func_to_call [str] function name to be called to convert the list of coverages computed for each bam file at each position into a single value. An example is a function that takes the ratio between the coverage of two bam files.

func_args [dict] dict of arguments to pass to *func*.

smoothLength [int] Distance in bp for smoothing the coverage per tile.

bed_regions_list: list List of tuples of the form (chrom, start, end) corresponding to bed regions to be processed. If not bed file was passed to the object constructor then this list is empty.

A list of [chromosome, start, end, temporary file], where the temporary file contains the bedgraph results for the region queried.

```
>>> test_path = os.path.dirname(os.path.abspath(__file__)) + "/test/test_data/
↪"
>>> bamFile1 = test_path +  "testA.bam"
>>> bin_length = 50
>>> number_of_samples = 0 # overruled by step_size
>>> func_to_call = scaleCoverage
>>> funcArgs = {'scaleFactor': 1.0}
```

```
>>> c = WriteBedGraph([bamFile1], bin_length, number_of_samples, stepSize=50)
>>> tempFile = c.writeBedGraph_worker( '3R', 0, 200, func_to_call, funcArgs)
>>> f = open(tempFile[3], 'r')
>>> f.readlines()
['3R\t0\t100\t0\n', '3R\t100\t200\t1\n']
>>> f.close()
>>> os.remove(tempFile[3])
```

`deeptools.writeBedGraph.`**`bedGraphToBigWig`**(*chromSizes*, *bedGraphFiles*, *bigWigPath*)
Takes a sorted list of bedgraph files and write them to a single bigWig file using pyBigWig. The order of bedGraphFiles must match that of chromSizes!

`deeptools.writeBedGraph.`**`getGenomeChunkLength`**(*bamHandles*, *tile_size*, *mappedList*)
Tries to estimate the length of the genome sent to the workers based on the density of reads per bam file and the number of bam files.

The chunk length should be a multiple of the tileSize

`deeptools.writeBedGraph.`**`ratio`**(*tile_coverage*, *args*)
tileCoverage should be an list of two elements

`deeptools.writeBedGraph.`**`scaleCoverage`**(*tile_coverage*, *args*)
tileCoverage should be an list with only one element

`deeptools.writeBedGraph.`**`writeBedGraph_wrapper`**(*args*)
Passes the arguments to writeBedGraph_worker. This is a step required given the constrains from the multiprocessing module. The args var, contains as first element the 'self' value from the WriteBedGraph object

deeptools.writeBedGraph_bam_and_bw module

`deeptools.writeBedGraph_bam_and_bw.`**`getCoverageFromBigwig`**(*bigwigHandle*, *chrom*, *start*, *end*, *tileSize*, *missingDataAsZero=False*) `deeptools.writeBedGraph_bam_and_bw.`**`writeBedGraph`**(*bamOrBwFileList*, *outputFileName*, *fragmentLength*, *func*, *funcArgs*, *tileSize=25*, *region=None*, *blackListFileName=None*, *numberOfProcessors=1*, *format='bedgraph'*, *extendPairedEnds=True*, *missingDataAsZero=False*, *skipZeroOverZero=False*, *smoothLength=0*, *fixed_step=False*, *verbose=False*)

Given a list of bamfiles, a function and a function arguments, this method writes a bedgraph file (or bigwig) file for a partition of the genome into tiles of given size and a value for each tile that corresponds to the given function and that is related to the coverage underlying the tile.

`deeptools.writeBedGraph_bam_and_bw.`**`writeBedGraph_worker`**(*chrom*, *start*, *end*, *tileSize*, *defaultFragmentLength*, *bamOrBwFileList*, *func*, *funcArgs*, *extendPairedEnds=True*, *smoothLength=0*, *skipZeroOverZero=False*, *missingDataAsZero=False*, *fixed_step=False*)

Writes a bedgraph having as base a number of bam files.

The given func is called to compute the desired bedgraph value using the funcArgs

tileSize

`deeptools.writeBedGraph_bam_and_bw.`**`writeBedGraph_wrapper`**(*args*)

Module contents

deepTools Galaxy.	code @ github.

Complete information can be found in the following links: genindex and modindex

deepTools Galaxy.	code @ github.

1.11 About

Please cite deepTools as follows:

> Ramírez, Fidel, Devon P. Ryan, Björn Grüning, Vivek Bhardwaj, Fabian Kilpert, Andreas S. Richter, Steffen Heyne, Friederike Dündar, and Thomas Manke. deepTools2: A next Generation Web Server for Deep-Sequencing Data Analysis. Nucleic Acids Research (2016). doi:10.1093/nar/gkw257.

Where deepTools are used:

- DEEP consortium
- public Galaxy server hosted at https://usegalaxy.org/.
- public Galaxy instance hosted by the Max-Planck-Institute for Immunobiology and Epigenetics: deeptools.ie-freiburg.mpg.de

- in-house Galaxy instance of the Max-Planck-Institute for Immunobiology and Epigenetics
- Galaxy instance of the University of Freiburg, Germany
- Galaxy instance of the ICGMB, Strasbourg, France
- Galaxy instance of LCSB and HPC @ Uni.lu, Belval, Luxembourg

MAX PLANCK INSTITUTE
OF IMMUNOBIOLOGY AND EPIGENETICS

This tool suite is developed by the Bioinformatics Facility at the Max Planck Institute for Immunobiology and Epigenetics, Freiburg.

deepTools Galaxy.	code @ github.

While developing deepTools, we continuously strive to create software that fulfills the following criteria:

- **efficiently extract reads from BAM files** and perform various computations on them
- **turn BAM files of aligned reads into bigWig files** using different normalization strategies
- make use of **multiple processors** (speed!)
- generation of **highly customizable images** (change colours, size, labels, file format, etc.)
- enable **customized down-stream analyses**, meaning that every data set created can be stored by the user
- **modular approach** - compatibility, flexibility, scalability (i.e. we can add more and more modules and make use of established methods)

Tip: For support or questions please post to Biostars. For bug reports and feature requests please open an issue <on github.

Please cite deepTools2 as follows:

Ramírez, Fidel, Devon P. Ryan, Björn Grüning, Vivek Bhardwaj, Fabian Kilpert, Andreas S. Richter, Steffen Heyne, Friederike Dündar, and Thomas Manke. **"deepTools2: a next generation web server for deep-sequencing data analysis." Nucleic Acids Research (2016): gkw257.**

MAX PLANCK INSTITUTE
OF IMMUNOBIOLOGY AND EPIGENETICS

This tool suite is developed by the Bioinformatics Facility at the Max Planck Institute for Immunobiology and Epigenetics, Freiburg.

deepTools Galaxy.	code @ github.

Python Module Index

d

`deeptools`, 104
`deeptools.bamHandler`, 83
`deeptools.correlation`, 84
`deeptools.correlation_heatmap`, 85
`deeptools.countReadsPerBin`, 86
`deeptools.deepBlue`, 92
`deeptools.getFragmentAndReadSize`, 93
`deeptools.getRatio`, 93
`deeptools.getScorePerBigWigBin`, 94
`deeptools.heatmapper`, 95
`deeptools.heatmapper_utilities`, 97
`deeptools.mapReduce`, 98
`deeptools.SES_scaleFactor`, 83
`deeptools.sumCoveragePerBin`, 99
`deeptools.utilities`, 100
`deeptools.writeBedGraph`, 101
`deeptools.writeBedGraph_bam_and_bw`, 103

Index

B

`bam_blacklisted_reads()` (*in module deeptools.utilities*), 100
`bam_blacklisted_worker()` (*in module deeptools.utilities*), 100
`bam_total_reads()` (*in module deeptools.utilities*), 100
`bedGraphToBigWig()` (*in module deeptools.writeBedGraph*), 103
`blSubtract()` (*in module deeptools.mapReduce*), 98

C

`change_chrom_names()` (*deeptools.heatmapper.heatmapper static method*), 95
`chopRegions()` (*in module deeptools.heatmapper*), 95
`chopRegionsFromMiddle()` (*in module deeptools.heatmapper*), 95
`chroms()` (*deeptools.deepBlue.deepBlue method*), 92
`close()` (*deeptools.deepBlue.deepBlue method*), 92
`compute_correlation()` (*deeptools.correlation.Correlation method*), 84
`compute_ratio()` (*in module deeptools.getRatio*), 93
`compute_sub_matrix_worker()` (*deeptools.heatmapper.heatmapper static method*), 96
`compute_sub_matrix_wrapper()` (*in module deeptools.heatmapper*), 95
`computeMatrix()` (*deeptools.heatmapper.heatmapper method*), 95
`computeSilhouetteScore()` (*in module deeptools.heatmapper*), 95
`convertCmap()` (*in module deeptools.utilities*), 100
`copyFileInMemory()` (*in module deeptools.utilities*), 100
`Correlation` (*class in deeptools.correlation*), 84
`count_reads_in_region()` (*deeptools.countReadsPerBin.CountReadsPerBin method*), 87
`countFragmentsInRegions_worker()` (*in module deeptools.getScorePerBigWigBin*), 94
`countReadsInInterval()` (*in module deeptools.bamHandler*), 83
`countReadsInRegions_wrapper()` (*in module deeptools.countReadsPerBin*), 91
`countReadsInRegions_wrapper()` (*in module deeptools.getScorePerBigWigBin*), 94
`CountReadsPerBin` (*class in deeptools.countReadsPerBin*), 86
`coverage_from_array()` (*deeptools.heatmapper.heatmapper static method*), 96
`coverage_from_big_wig()` (*deeptools.heatmapper.heatmapper static method*), 96

D

`deepBlue` (*class in deeptools.deepBlue*), 92
`deeptools` (*module*), 104
`deeptools.bamHandler` (*module*), 83
`deeptools.correlation` (*module*), 84
`deeptools.correlation_heatmap` (*module*), 85
`deeptools.countReadsPerBin` (*module*), 86
`deeptools.deepBlue` (*module*), 92
`deeptools.getFragmentAndReadSize` (*module*), 93
`deeptools.getRatio` (*module*), 93
`deeptools.getScorePerBigWigBin` (*module*), 94
`deeptools.heatmapper` (*module*), 95
`deeptools.heatmapper_utilities` (*module*), 97
`deeptools.mapReduce` (*module*), 98
`deeptools.SES_scaleFactor` (*module*), 83
`deeptools.sumCoveragePerBin` (*module*), 99
`deeptools.utilities` (*module*), 100
`deeptools.writeBedGraph` (*module*), 101

deeptools.writeBedGraph_bam_and_bw (*module*), 103

E

estimateScaleFactor() (*in module deeptools.SES_scaleFactor*), 83

estimateSizeFactors() (*in module deeptools.countReadsPerBin*), 91

G

get_chunk_length() (*deeptools.countReadsPerBin.CountReadsPerBin method*), 89

get_coverage_of_region() (*deeptools.countReadsPerBin.CountReadsPerBin method*), 89

get_coverage_of_region() (*deeptools.sumCoveragePerBin.SumCoveragePerBin method*), 100

get_fragment_from_read() (*deeptools.countReadsPerBin.CountReadsPerBin method*), 89

get_individual_matrices() (*deeptools.heatmapper.heatmapper method*), 96

get_num_individual_matrix_cols() (*deeptools.heatmapper.heatmapper method*), 96

get_outlier_indices() (*deeptools.correlation.Correlation static method*), 84

get_read_and_fragment_length() (*in module deeptools.getFragmentAndReadSize*), 93

getChroms() (*deeptools.deepBlue.deepBlue method*), 92

getChromSizes() (*in module deeptools.getScorePerBigWigBin*), 94

getCommonChrNames() (*in module deeptools.utilities*), 100

getCoverageFromBigwig() (*in module deeptools.writeBedGraph_bam_and_bw*), 103

getEID() (*deeptools.deepBlue.deepBlue method*), 92

getFragmentLength_worker() (*in module deeptools.getFragmentAndReadSize*), 93

getFragmentLength_wrapper() (*in module deeptools.getFragmentAndReadSize*), 93

getGC_content() (*in module deeptools.utilities*), 100

getGenome() (*deeptools.deepBlue.deepBlue method*), 92

getGenomeChunkLength() (*in module deeptools.writeBedGraph*), 103

getMappingStats() (*in module deeptools.bamHandler*), 83

getProfileTicks() (*in module deeptools.heatmapper_utilities*), 97

getRatio() (*in module deeptools.getRatio*), 93

getRead() (*deeptools.countReadsPerBin.Tester method*), 91

getReadLength() (*deeptools.countReadsPerBin.CountReadsPerBin method*), 88

getScorePerBin() (*in module deeptools.getScorePerBigWigBin*), 94

getSmoothRange() (*deeptools.countReadsPerBin.CountReadsPerBin method*), 88

getTempFileName() (*in module deeptools.utilities*), 100

getTicks() (*deeptools.heatmapper.heatmapper method*), 96

getTLen() (*in module deeptools.utilities*), 100

getUserRegion() (*in module deeptools.mapReduce*), 98

gtfOptions() (*in module deeptools.utilities*), 100

H

heatmapper (*class in deeptools.heatmapper*), 95

I

is_proper_pair() (*deeptools.countReadsPerBin.CountReadsPerBin static method*), 90

isDeepBlue() (*in module deeptools.deepBlue*), 92

L

load_matrix() (*deeptools.correlation.Correlation method*), 85

M

makeChromTiles() (*in module deeptools.deepBlue*), 92

makeRegions() (*in module deeptools.deepBlue*), 92

makeTiles() (*in module deeptools.deepBlue*), 92

mapReduce() (*in module deeptools.mapReduce*), 98

matrix_avg() (*deeptools.heatmapper.heatmapper static method*), 96

matrix_from_dict() (*deeptools.heatmapper.heatmapper method*), 96

mergeRegions() (*in module deeptools.deepBlue*), 92

mungeChromosome() (*in module deeptools.utilities*), 100

my_average() (*deeptools.heatmapper.heatmapper static method*), 96

O

openBam() (*in module deeptools.bamHandler*), 84

P

`plot_correlation()` (*deeptools.correlation.Correlation method*), 85
`plot_correlation()` (*in module deeptools.correlation_heatmap*), 85
`plot_pca()` (*deeptools.correlation.Correlation method*), 85
`plot_scatter()` (*deeptools.correlation.Correlation method*), 85
`plot_single()` (*in module deeptools.heatmapper_utilities*), 97
`plotly_correlation()` (*deeptools.correlation.Correlation method*), 85
`plotly_pca()` (*deeptools.correlation.Correlation method*), 85
`plotly_scatter()` (*deeptools.correlation.Correlation method*), 85
`plotly_single()` (*in module deeptools.heatmapper_utilities*), 98
`preload()` (*deeptools.deepBlue.deepBlue method*), 92
`preloadWrapper()` (*in module deeptools.deepBlue*), 92

R

`ratio()` (*in module deeptools.writeBedGraph*), 103
`read_matrix_file()` (*deeptools.heatmapper.heatmapper method*), 96
`remove_outliers()` (*deeptools.correlation.Correlation method*), 85
`remove_row_of_zeros()` (*in module deeptools.countReadsPerBin*), 92
`remove_rows_of_zeros()` (*deeptools.correlation.Correlation method*), 85
`run()` (*deeptools.countReadsPerBin.CountReadsPerBin method*), 91
`run()` (*deeptools.writeBedGraph.WriteBedGraph method*), 102

S

`save_BED()` (*deeptools.heatmapper.heatmapper method*), 96
`save_corr_matrix()` (*deeptools.correlation.Correlation method*), 85
`save_matrix()` (*deeptools.heatmapper.heatmapper method*), 96
`save_matrix_values()` (*deeptools.heatmapper.heatmapper method*), 96
`save_tabulated_values()` (*deeptools.heatmapper.heatmapper method*), 97
`scaleCoverage()` (*in module deeptools.writeBedGraph*), 103
`smartLabel()` (*in module deeptools.utilities*), 101
`smartLabels()` (*in module deeptools.utilities*), 101
`SumCoveragePerBin` (*class in deeptools.sumCoveragePerBin*), 99

T

`tbitToBamChrName()` (*in module deeptools.utilities*), 101
`Tester` (*class in deeptools.countReadsPerBin*), 91
`Tester` (*class in deeptools.getScorePerBigWigBin*), 94
`Tester` (*class in deeptools.SES_scaleFactor*), 83
`Tester` (*class in deeptools.sumCoveragePerBin*), 100
`toBytes()` (*in module deeptools.utilities*), 101
`toString()` (*in module deeptools.utilities*), 101
`trimZones()` (*in module deeptools.heatmapper*), 97

W

`WriteBedGraph` (*class in deeptools.writeBedGraph*), 101
`writeBedGraph()` (*in module deeptools.writeBedGraph_bam_and_bw*), 103
`writeBedGraph_worker()` (*deeptools.writeBedGraph.WriteBedGraph method*), 102
`writeBedGraph_worker()` (*in module deeptools.writeBedGraph_bam_and_bw*), 104
`writeBedGraph_wrapper()` (*in module deeptools.writeBedGraph*), 103
`writeBedGraph_wrapper()` (*in module deeptools.writeBedGraph_bam_and_bw*), 104