gamtools Documentation

Release 0.1

Rob Beagrie

May 06, 2017

Contents

GAMtools is a collection of utilities for working with Genome Architecture Mapping data. GAM is a technique for mapping 3D genome architecture by sequencing genomic DNA from thin nuclear sections (nuclear profiles or NPs). GAMtools can be used to automate the mapping and processing of sequencing data from NPs, to identify genomic regions present in each NP, and to calculate proximity matrices based on the co-segregation of genomic regions in a dataset of many NPs.

CHAPTER 1

Tutorial

The GAMtools tutorial covers some of the basic use cases for GAMtools, and will guide you through re-mapping and re-processing of some example GAM data.

Overview

Summary of available tools.

GAMtools provides a number of different utilities for working with GAM data. The table below summarizes the tools available in the suite.

Utility	Description
call_windows	Call positive windows for individual NPs
compaction	Calculate chromatin compaction
convert	Convert between different GAM matrix formats
enrichment	Calculate enrichments of SLICE interactions
matrix	Generate a GAM matrix from a segregation file
permute_segregation	Circularly permute the columns of a GAM segregation file
plot_np	Plot the segregation results for a particular NP
process_nps	Map raw GAM sequencing data and call positive windows
radial_pos	Calculate chromatin radial position
select	Select only certain samples from a segregation file

Installation

GAMtools is intended to run in a command line environment on UNIX, LINUX and Apple OS X operating systems. GAMtools can also be installed on Windows using cygwin.

The recommended way to install **GAMtools** is to use python's package manager, `pip`. This method should ensure that all of GAMtools required dependencies are installed automatically.

Alternatively, **GAMtools** can be installed by downloading the source code and compiling it manually.

Installing stable releases using pip

To install the latest stable release using pip, you need to run the following command:

```
$ pip install gamtools
```

pip should automatically find and install any *mandatory dependencies* that are not currently installed. Additional *optional dependencies* are required for full **GAMtools** functionality, but these must be installed manually.

Installing from source (GitHub)

If you want to install the latest development version of **GAMtools**, you will need to install from source code. First, clone the GAMtools repository from GitHub:

```
$ git clone https://github.com/pombo-lab/gamtools.git
```

Then install the downloaded package using pip:

```
$ pip install gamtools/
```

Or if pip is not installed:

```
$ cd gamtools
$ python setup.py install
```

Installation using pip is the preferred method, as this will handle installing the *mandatory dependencies* automatically. If **GAMtools** is installed using `python setup.py install` you may need to manually install *mandatory dependencies* yourself.

Troubleshooting

GAMtools requires numpy and cython to be installed before it can compile properly. If you are installing using `pip`, numpy and cython should be installed automatically, but there is a chance this might not work. If you are having issues installing **GAMtools**, the first step is to ensure both numpy and cython are properly installed:

```
$ pip install cython numpy
```

If you are still having problems, please post a ticket on our GitHub issues page.

Mandatory dependencies

GAMtools depends on a number of additional python libraries, which must be installed for it to function correctly. These libraries are normally installed automatically during the **GAMtools** installation process.

Mandatory python dependencies

- doit
- numpy

- scipy
- cython
- pandas
- wrapit

These python libraries can all be installed using pip:

```
$ pip install doit numpy scipy cython pandas wrapit
```

Optional dependencies

Some features in **GAMtools** depend on additional libraries and/or programs which are not installed automatically.

Making plots

The `gamtools plot_np` command requires some python plotting libraries to be installed. These may also be required for the `gamtools call_windows` command if the `--fitting-folder` flag is specified.

Optional python dependencies

- matplotlib
- pybedtools
- metaseq

Working with raw sequencing data

The `gamtools process_nps` command is used to map and process raw sequencing data from NPs. This can require a number of additional command line programs to be installed and configured:

Mapping and processing programs

Program	Required for
Bowtie2	Mapping raw sequencing data.
samtools	Mapping raw sequencing data.
bedtools	Calling positive windows for an NP.
bedGraphToBigWig	Creating bigwigs (`--bigwigs` flag)
bedToBigBed	Creating bigbeds (`--bigbeds` flag)
fastqc	Performing dataset quality control (`--do-qc` flag)
fastq_screen	Performing dataset quality control (`--do-qc` flag)

Testing your installation

To test that you have installed gamtools and all its dependencies correctly you can run the command `gamtools test`. If you have skipped installing any optional dependencies, you may get a warning message saying something like "x could not be found, and is required for y". You can safely ignore these messages unless you need the particular gamtools functionality in the message.

Tutorial

First steps

Installing GAMtools

The first step in the **GAMtools** tutorial is to make sure that **GAMtools** is properly installed. Try to run `gamtools --help` and make sure that you get the following ouput:

```
$ gamtools --help
usage: gamtools [-h]
                {call_windows,convert,enrichment,matrix,permute_segregation,plot_np,
↪process_nps,select}
                ...
```

If this command gives you an error message, it is likely that **GAMtools** has not been installed correctly. Please ensure you have followed the steps outlined in the *Installation* guide.

Downloading the tutorial data

Once **GAMtools** is working correctly, you need to download some example data to work with during the tutorial. The tutorial data is located on the GAMtools website. Download the tutorial data (e.g. by using wget), extract it and cd into the newly created directory. The directory should contain a folder called `fastqs` and a file called *clean.sh*.

```
$ wget http://gam.tools/tutorial_data.tar.gz
$ tar zxvf tutorial_data.tar.gz
$ cd gamtools_tutorial
$ ls
clean.sh  fastqs/
```

The `fastqs` folder contains sequencing data from 100 separate nuclear profiles (NPs):

```
$ ls fastqs/
NP_001.fq.gz  NP_026.fq.gz  NP_051.fq.gz  NP_076.fq.gz
NP_002.fq.gz  NP_027.fq.gz  NP_052.fq.gz  NP_077.fq.gz
NP_003.fq.gz  NP_028.fq.gz  NP_053.fq.gz  NP_078.fq.gz
NP_004.fq.gz  NP_029.fq.gz  NP_054.fq.gz  NP_079.fq.gz
NP_005.fq.gz  NP_030.fq.gz  NP_055.fq.gz  NP_080.fq.gz
...
NP_025.fq.gz  NP_050.fq.gz  NP_075.fq.gz  NP_100.fq.gz
```

These files are the primary raw output of a GAM experiment. The first thing we need to do with the sequencing data is to "map" it to a genome. The example data comes from mouse embryonic stem cells, so we need to map it to the mouse genome, which we will do using bowtie2. If you already have bowtie2 and a mouse genome assembly installed and configured on your local machine, you can skip the next step (mouse assembly mm9 is preferred, but any other assembly should work with this tutorial).

Configuring bowtie2

If you have not yet installed bowtie2, please follow the installation instructions on the bowtie2 homepage. Once you have bowtie installed, verify that everything is working correctly:

```
$ bowtie2 --version
/home/rob_000/bowtie2-2.2.9/bowtie2-align-s version 2.2.9
64-bit
Built on Windows8
30 Apr 2016 18:13:39
```

We next need to provide the sequence of the mouse genome for bowtie to map against. If you wish, you can download and configure the full mouse mm9 "index" from Illumina. However, the 100 sequencing datasets provided as part of the tutorial only contain sequencing data from a small region of chromosome 19, so you can also use a special truncated index containing only the sequence of mouse chromosome 19. This will allow bowtie to run much faster whilst using less RAM, and is perfectly sufficient for completing this tutorial. If you wish to use the tutorial index, download it from the **GAMtools** website, extract it to the same folder as fastqs and configure bowtie to use the new truncated index:

```
$ wget http://gam.tools/tutorial_index.tar.gz
$ tar zxvf tutorial_index.tar.gz
$ ls
clean.sh fastqs/ genome/
$ export BOWTIE2_INDEXES=$(pwd)/genome/
$ ls $BOWTIE2_INDEXES
genome.1.bt2  genome.3.bt2  genome.rev.1.bt2  chr19.size
genome.2.bt2  genome.4.bt2  genome.rev.2.bt2
```

Mapping the sequencing data and calling positive windows

The **GAMtools** command used for mapping NP sequencing data is `gamtools process_nps`. The `process_nps` command has a lot of different parameters and options, you can use the `--help` flag to get a full description of all the available parameters. Further information about the `process_nps` command can also be found on the *process_nps* page.

```
$ gamtools process_nps --help
usage: gamtools process_nps [-h] -g GENOME_FILE [-o OUPUT_DIRECTORY]
                            [-f FITTINGS_DIRECTORY] [-d DETAILS_FILE] [-i]
                            [-b] [-c] [-w WINDOW_SIZE [WINDOW_SIZE ...]] [-m]
                            [-s MATRIX_SIZE [MATRIX_SIZE ...]]
                            [--qc-window-size QC_WINDOW_SIZE]
                            [--additional-qc-files [ADDITIONAL_QC_FILES [ADDITIONAL_
↪QC_FILES ...]]]
                            [-q MINIMUM_MAPQ] [--doit-db-file DEP_FILE]
                            [--doit-backend {sqlite3,json,dbm}]
                            [--doit-verbosity {0,1,2}]
                            [--doit-reporter {json,console,zero,executed-only}]
                            [--doit-process NUM_PROCESS]
                            [--doit-parallel-type {process,thread}]
                            INPUT_FASTQ [INPUT_FASTQ ...]
```

For now, we can just use the default options. That means that all we need to specifiy is a genome file (using `-g/--genome-file`) and a list of input fastq files:

```
$ gamtools process_nps -g genome/chr19.size fastqs/*.fq.gz
```

This tells **GAMtools** to use the genome file `genome/chr19.size` .You will have this file if you downloaded the special truncated index. If you are using your own mouse genome index, you will have to specify your own genome file (which is usually named something like `mm9.chrom.sizes`). The next argument tells **GAMtools** to process all

of the files with the extension ".fq.gz" in the folder called "fastqs". When you run the command, **GAMtools** will start mapping the sequencing data, and you should see an output like this:

```
$ gamtools process_nps -g genome/chr19.size fastqs/*.fq.gz
-- Creating output directory
.  Mapping fastq:fastqs/NP_025.fq.gz
.  Mapping fastq:fastqs/NP_017.fq.gz
.  Mapping fastq:fastqs/NP_065.fq.gz
.  Mapping fastq:fastqs/NP_014.fq.gz
.  Mapping fastq:fastqs/NP_090.fq.gz
.  Mapping fastq:fastqs/NP_078.fq.gz
```

GAMtools will then proceed to map all 100 individual sequencing files to the mouse genome. This will take around 5 minutes if you are using the truncated index and a moderately fast computer. If you are using your own full mouse genome index, it may take a little longer. Once it has mapped the files, **GAMtools** will sort the mapped files, remove PCR duplicates and create an index for fast data retrieval.

The final steps are to compute the number of reads from each NP that overlap each 50kb window in the supplied genome file, and then to use this read coverage count to determine which of the windows was present in the original NP. After performing this "window calling" step, gamtools produces a file called `segregation_at_50kb.table`. This file contains one row per 50kb window, and one column per NP:

```
# Show the first 10 rows and first 5 columns of the segregation table
$ head segregation_at_50kb.table | cut -f 1-5
chrom   start   stop    fastqs/NP_027.rmdup.bam       fastqs/NP_020.rmdup.bam
chr19   0       50000   0       0
chr19   50000   100000  0       0
chr19   100000  150000  0       0
chr19   150000  200000  0       0
chr19   200000  250000  0       0
chr19   250000  300000  0       0
chr19   300000  350000  0       0
chr19   350000  400000  0       0
chr19   400000  450000  0       0
```

For each NP column, `0` indicates that the window was not present in the NP, whereas `1` indicates that the window was present. This table is the crucial and most important output of a GAM experiment - all further downstream analysis will generally be based on the segregation table.

Producing proximity matrices

Now that we have produced a segregation table at 50kb resolution, we can use it to calculate a proximity matrix, using the `gamtools matrix` command. As for the *process_nps* command, the matrix command has a lot of different options, which can be explored further using the `--help` flag or on the *gamtools matrix* page.

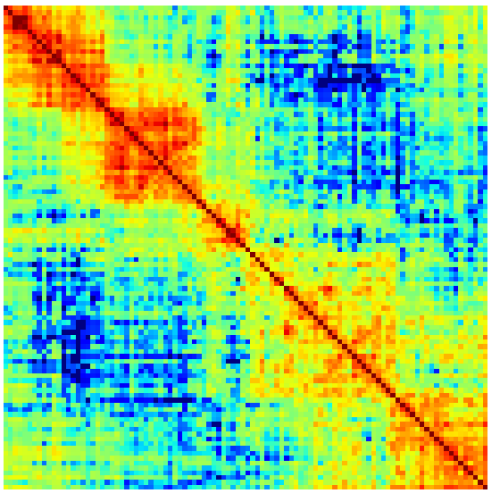
```
$ gamtools matrix --help
usage: gamtools matrix [-h] -r REGION [REGION ...] -s SEGREGATION_FILE
                       [-f {csv.gz,txt,csv,txt.gz,npz}]
                       [-t {cosegregation,linkage,dprime}] [-o OUTPUT_FILE]

optional arguments:
  -h, --help            show this help message and exit
  -r REGION [REGION ...], --regions REGION [REGION ...]
                        Specific genomic regions to calculate matrices for. If
                        one region is specified, a matrix is calculated for
                        that region against itself. If more than one region is
                        specified, a matrix is calculated for each region
```

```
                        against the other. Regions are specified using UCSC
                        browser syntax, i.e. "chr4" for the whole of
                        chromosome 4 or "chr4:100000-200000" for a sub-region
                        of the chromosome.
  -s SEGREGATION_FILE, --segregation_file SEGREGATION_FILE
                        A segregation file to use as input
  -f {csv.gz,txt,csv,txt.gz,npz}, --output-format {csv.gz,txt,csv,txt.gz,npz}
                        Output matrix file format (choose from: csv.gz, txt,
                        csv, txt.gz, npz, default is txt.gz)
  -t {cosegregation,linkage,dprime}, --matrix-type {cosegregation,linkage,dprime}
                        Method used to calculate the interaction matrix
                        (choose from: cosegregation, linkage, dprime, default
                        is dprime)
  -o OUTPUT_FILE, --output-file OUTPUT_FILE
                        Output matrix file. If not specified, new file will
                        have the same name as the segregation file and an
                        extension indicating the genomic region(s) and the
                        matrix method
```

We can start by asking for the proximity matrix for our region of interest in png format:

```
$ gamtools matrix -s segregation_at_50kb.table \
> -r chr19:10,000,000-15,000,000 -o my_matrix.png
starting calculation for chr19:10,000,000-15,000,000
region size is: 100 x 100 Calculation took 1.05s
Saving matrix to file my_matrix.png
Done!
$ open my_matrix.png
```

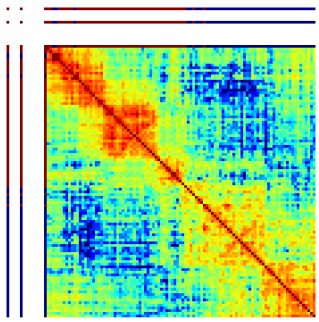
You should see an image file that looks like this:



Note that the example data for this tutorial only covers this specific region of chromosome 19, so if you specify a larger or different region you will get some strange looking results:

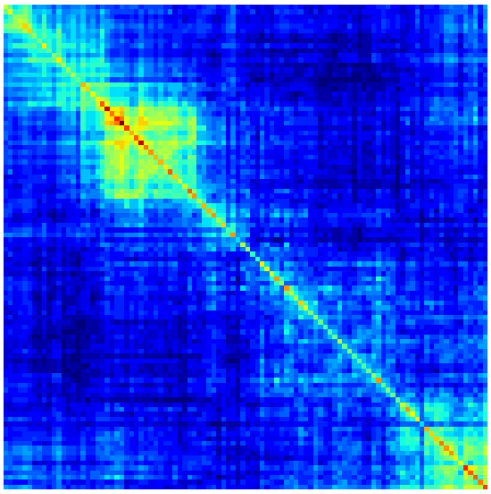
```
$ gamtools matrix -s segregation_at_50kb.table \
> -r chr19:8,000,000-17,000,000 -o larger_matrix.png
```

```
starting calculation for chr19:8,000,000-17,000,000
region size is: 180 x 180 Calculation took 3.47s
Saving matrix to file larger_matrix.png
Done!
$ open larger_matrix.png
```



By default, **GAMtools** produces proximity matrices using the normalized linkage disequilibrium (or **D'**). In this case, it first calculates how many times each pair of windows are found together in the same NP, and then normalizes the matrix according to how many times each window is detected across the collection of NPs. You can create raw, unnormalized co-segregation matrices by specifying the `cosegregation` option using the `-t/--matrix-type` flag:

```
$ gamtools matrix -s segregation_at_50kb.table \
> -r chr19:10,000,000-15,000,000 -o cosegregation_matrix.png \
> -t cosegregation
starting calculation for chr19:10,000,000-15,000,000
region size is: 100 x 100 Calculation took 1.05s
Saving matrix to file cosegregation_matrix.png
Done!
$ open cosegregation_matrix.png
```



Working at different resolutions

If we want to produce a proximity matrix at a resolution other than 50kb, we first need to calculate a segregation table at that resolution. We can generate another segregation table using the *process_nps* command, specifying the resolution using the `-w/--window-sizes` flag. For example at 30kb resolution:

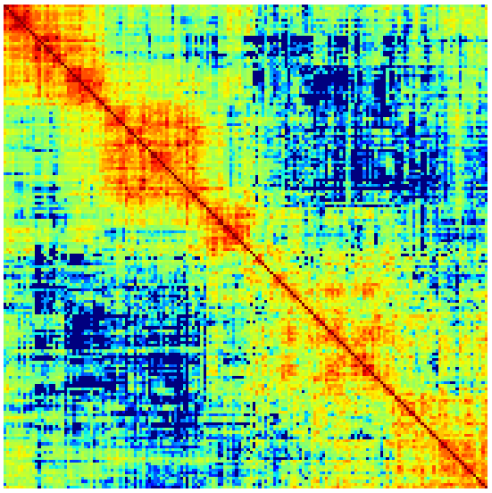
```
$ gamtools process_nps -w 30000 -g genome/chr19.size fastqs/*.fq.gz
-- Creating output directory
-- Mapping fastq:fastqs/NP_025.fq.gz
-- Mapping fastq:fastqs/NP_017.fq.gz
-- Mapping fastq:fastqs/NP_065.fq.gz
-- Mapping fastq:fastqs/NP_014.fq.gz
-- Mapping fastq:fastqs/NP_090.fq.gz
-- Mapping fastq:fastqs/NP_078.fq.gz
...
...
...
.  Getting coverage:30kb windows
.  Calling positive windows:30kb
```

Notice that all the lines except the last two begin with `--`, whereas the last two lines begin with `.`. The `--` indicates that **GAMtools** realized that these tasks have already been completed and therefore do not need to be re-run. When we re-calculate a segregation table at a new resolution, we don't need to remap all the individual fastq files, we only need to re-compute the read depth over all 30kb windows, and then decide which 30kb windows were positive in each NP.

To create proximity matrices at the new resolution, we need to specify the new segregation table: `segregation_at_30kb.table`.

```
$ gamtools matrix -s segregation_at_30kb.table \
> -r chr19:10,000,000-15,000,000 -o 30kb_matrix.png
starting calculation for chr19:10,000,000-15,000,000
region size is: 167 x 167 Calculation took 0.047s
Saving matrix to file 30kb_matrix.png
```

```
Done!
$ open 30kb_matrix.png
```



Performing quality control checks

If you are generating your own GAM datasets, you will want to perform some checks to ensure your NPs are of sufficient quality. **GAMtools** will generate a table of QC parameters automatically for each NP if you use the *process_nps* command with the `-c/--do-qc` flag.

Note: Performing quality control requires a number of additional dependencies to be installed. Please ensure that `gamtools test` runs with no errors before continuing with this section.

Re-running the `gamtools process_nps` command with the `--do-qc` flag will instruct **GAMtools** to run a number of additional tasks. Your output should look something like this:

```
$ gamtools process_nps --do-qc -g genome/chr19.size fastqs/*.fq.gz
-- Creating output directory
-- Mapping fastq:fastqs/NP_025.fq.gz
-- Mapping fastq:fastqs/NP_017.fq.gz
-- Mapping fastq:fastqs/NP_065.fq.gz
...
...
...
.  Creating QC parameters file with default values
.  Getting mapping stats
.  Getting segregation stats
.  Running fastqc:fastqs/NP_042.fq.gz
.  Running fastqc:fastqs/NP_043.fq.gz
...
.  Running fastqc:fastqs/NP_070.fq.gz
.  Running fastq_screen:fastqs/NP_063.fq.gz
.  Running fastq_screen:fastqs/NP_050.fq.gz
```

```
...
.  Running fastq_screen:fastqs/NP_081.fq.gz
.  Getting quality stats
.  Getting contamination stats
.  Merging stats files
.  Finding samples that pass QC
.  Filtering samples based on QC values:50kb
```

By default, GAMtools generates several QC files, each containing different information about the collection of NPs:

- The number of sequenced, mapped, and unique (i.e. excluding PCR duplicates) reads are saved in `mapping_stats.txt`
- Statistics regarding the number and distribution of positive windows are saved in `segregation_stats.txt`
- Statistics regarding the sequencing quality scores and the number of mono- and di-nucleotide repeat containing reads are calculated by fastqc and saved to `quality_stats.txt`
- Statistics regarding the percentage of reads mapping to different genomes (i.e. contaminating reads) are calculated by fastq_screen and saved to `contamination_stats.txt`
- These statistics files are merged together and the resulting table containing all the different QC parameters is saved to `merged_stats.txt`

Once the merged stats table has been saved, **GAMtools** will attempt to filter out "poor quality" NPs, and generates a file called `samples_passing_qc.txt` containing only high-quality NPs. **GAMtools** filters out NPs which match any rules in the `qc_parameters.cfg` file, which is created with some default rules if it does not exist. Finally, **GAMtools** creates new segregation tables that exclude poor-quality NPs. In our case, this file will be called `segregation_at_50kb.passed_qc.table`. You can use this new segregation table to re-generate the proximity matrices (see *Producing proximity matrices*).

gamtools API

Background

GAMtools offers a programmatic API in python, which allows other programmers or bioinformaticians to use GAMtools functionality in their own applications or pipelines.

gamtools.call_windows module

gamtools.compaction module

gamtools.cosegregation module

gamtools.enrichment module

gamtools.matrix module

gamtools.permutation module

gamtools.radial_position module

gamtools.segregation module

call_windows

The `gamtools call_windows` tool determines which genomic regions were present in each NP. The input file is a tab delimited table in which the first three columns indicate the genomic region in bed format (chrom, start, stop) and the remaining columns give the number of reads mapping to each genomic region for each NP. For example:

```
chrom   start    stop     NP_1   NP_2   NP_3   NP_4   NP_5
chr19   0        50000    0      0      0      0      0
chr19   50000    100000   1      26     1      54     0
chr19   100000   150000   0      34     0      0      1
chr19   150000   200000   2      16     0      0      0
chr19   200000   250000   0      1      32     7      0
chr19   250000   300000   3      0      0      0      1
chr19   300000   350000   1      4      50     12     0
chr19   350000   400000   2      3      32     1      3
chr19   400000   450000   0      0      115    0      0
```

This type of coverage table can be generated quite easily using bedtools multicov and is generated automatically by the GAMtools *process_nps* command.

The output file is in the same format, but each entry is either a 1 (indicating the region was present) or a 0 (indicating that it was absent).

```
chrom   start    stop     NP_1   NP_2   NP_3   NP_4   NP_5
chr19   0        50000    0      0      0      0      0
chr19   50000    100000   0      1      0      1      0
chr19   100000   150000   0      1      0      0      0
chr19   150000   200000   0      1      0      0      0
chr19   200000   250000   0      0      1      0      0
chr19   250000   300000   0      0      0      0      0
chr19   300000   350000   0      0      1      1      0
chr19   350000   400000   0      0      1      0      0
chr19   400000   450000   0      0      1      0      0
```

Usage and option summary

Usage:

```
gamtools call_windows [OPTIONS] <COVERAGE_TABLE>
```

Optional parameters:

Option	Description
-d, –details-file	Write a table of fitting parameters to this path
-o, –output-file	Output segregation file to create (or "-" to write to stdout), default is stdout
-f, –fitting-folder	Save plots for each individual curve fitting to this folder

compaction

The `gamtools compaction` tool is used to calculate chromatin compaction from GAM segregation tables. Chromatin compaction is estimated from the number of NPs that contain a given chromatin region, since chromatin that

occupies a larger volume will be intersected by a greater number of NPs.

Usage and option summary

Usage:

```
gamtools compaction [OPTIONS] -s <SEGREGATION_FILE> -o <OUTPUT_FILE>
```

Optional parameters:

Option	Description
-n, –no-blanks	Exclude regions that were never detected from the output (for making bedgraphs)

convert

The `gamtools convert` tool is used to convert proximity matrices output by GAMtools to/from various different formats.

Usage and option summary

Usage:

```
gamtools convert [OPTIONS] <INPUT_MATRIX> <OUTPUT_MATRIX>
```

Optional parameters:

Option	Description
-i, –input-format	Input matrix file format
-o, –output-format	Output matrix file format
-t, –thresholds-file	Thresholds file. If specified, any values lower than the specified thresholds will be masked/excluded from the output file
-w, –windows-file	File containing the genomic locations of matrix bins (only required if not specified in input matrix file).
-r, –region	Region covered by the input matrix (required if -w /–windows-file is specified)

enrichment

The `gamtools enrichment` tool is used to calculate the enrichment of pairwise interactions between windows of different classes. For example, it can be used to answer the question of whether a particular set of interactions connects windows that contain genes with windows that contain enhancers more or less frequently than would be expected by chance.

`gamtools enrichment` requires two input files. The first is a tab-delimited table giving the pairwise interactions. This table must contain the following columns:

Column	Description
chrom	Name of the chromosome
Pos_A	index of the window on the left of the interaction
Pos_B	index of the window on the right of the interaction
interaction	strength of the interaction

Window indices used for **Pos_A** and **Pos_B** are 0-based, such that `0` would be the first window on the chromosome and `20` would be the 19th. An example file might look like:

```
chrom    Pos_A  Pos_B    interaction
chr1     10     20       0.75
chr2     10     20       0.50
chr1     10     30       0.40
```

The second input file is a comma-delimited (csv) table giving the classes of each window. This table can have the following columns:

Column	Description
chrom	Name of the chromosome
i	index of the window
start	Start co-ordinate of the window *(optional)*
stop	Stop co-ordinate of the window *(optional)*

Any additional columns will be interpreted as different classes. Each class column should indicate whether the given window is a member of the class with the values `False` or `True`. An example classification table might look like this:

```
chrom,i,Enhancer,Gene
chr1,10,True,False
chr1,20,False,True
chr2,20,True,False
chr2,30,False,True
```

The output is a csv file in the following format:

Column	Description
class1	First class involved in interaction
class2	Second class involved in interaction
count	Number of interactions where a window in **class1** interacts with a window in **class2**
permuted	Whether or not the interactions table was randomly permuted before counting

For example:

```
class1,class2,count,permuted
Gene,Gene,234148,yes
Gene,Enhancer,268228,yes
Enhancer,Enhancer,10598,yes
```

Usage and option summary

Usage:

```
gamtools enrichment [OPTIONS] -i <INTERACTIONS_FILE> -c <CLASSES_FILE>
```

Optional parameters:

Option	Description
-o, –output-prefix	First part of the output file name (default is “enrichment_results”)
-p, –permutations*	Number of times to randomly permute the input file
-n, –no-permute*	Do not permute the input file, instead calculate observed counts

* Options -p/-n are mutually exclusive, exactly one of these two options **must** be given

matrix

The `gamtools matrix` tool is used to calculate proximity matrices from segregation tables.

Usage and option summary

Usage:

```
gamtools matrix [OPTIONS] -s <SEGREGATION_FILE> -r <REGION> [<REGION> ...]
```

Optional parameters:

Option	Description
-f, –output-format	Output matrix file format (choose from: csv.gz, txt.gz, npz, txt, csv, png, default is txt.gz)
-t, –matrix-type	Method used to calculate the interaction matrix (choose from: cosegregation, linkage, dprime, default is dprime)
-o, –output-file	Output matrix file. If not specified, new file will have the same name as the segregation file and an extension indicating the genomic region(s) and the matrix method

Specifying regions

The -r/–regions parameter allows the user to specify the specific genomic regions to calculate matrices for. If one region is specified, a matrix is calculated for that region against itself. If more than one region is specified, a matrix is calculated for each region against the other. Regions are specified using UCSC browser syntax, i.e. "chr4" for the whole of chromosome 4 or "chr4:100000-200000" for a sub-region of the chromosome.

permute_segregation

The `gamtools permute_segregation` tool is used to circularly permute segregation tables. This can be handy for generating random background matrices, as the random permutation should remove any specific long-range interactions.

Usage and option summary

Usage:

```
gamtools permute_segregation -s <SEGREGATION_FILE> -o <OUTPUT_FILE>
```

plot_np

The `gamtools plot_np` tool is used to plot the sequencing coverage and the window calling output for a single NP over the whole genome. It can be very useful as a quick quality control, as NPs that have been sequenced correctly display a characteristic pattern of coverage, with many large regions (up to whole chromosomes) displaying very low coverage.

Plotting the coverage for an NP requires three files, a bigwig file made from the raw sequencing data, a bed file containing the positive windows and a file containing a list of chromosomes and their lengths in bp (a genome file).

Usage and option summary

Usage:

```
gamtools plot_np -w BIGWIG_FILE -b BED_FILE
                 -g GENOME_FILE -o OUTPUT_FILE
```

process_nps

The `gamtools process_nps` tool is used to map raw sequencing data from a collection of NPs and call positive windows from those NPs to generate a segregation table. It can optionally also calculate various QC metrics for each NP, generate bigwig/bed files for visualising the raw data and calculate proximity matrices.

Usage and option summary

Usage:

```
gamtools process_nps [OPTIONS] -g <GENOME_FILE> <FASTQ_FILE> [<FASTQ_FILE> ...]
```

Optional parameters:

Option	Description
-o, –output_dir	Write segregation, matrix etc. to this directory
-q, –minimum-mapq	Filter out any mapped read with a mapping quality less than x (default is 20, use -q 0 for no filtering)
-c, –do-qc	Perform sample quality control.
-i, –bigwigs	Make bigWig files.
-b, –bigbeds	Make bed files of positive windows
-w, –window-sizes	One or more window sizes for calling positive windows
-s, –matrix-sizes	Resolutions for which proximity matrices should be produced.
–qc-window-size	Use this window size for qc (default is median window size).
-f, –fittings_dir	Write segregation curve fitting plots to this directory
-d, –details-file	If specified, write a table of fitting parameters to this path
–additional-qc-files	Any additional qc files to filter on

Parameters inherited from doit:

`gamtools process_nps` uses doit as a task dependency engine, to determine what actions need to be performed and in which order. A number of additional command line parameters are available that control doit's behaviour.

Option	Description
–doit-db-file	Doit saves information about each run in a database file. This parameter specifies the location of that database file.
–doit-backend	Doit database format. (one of sqlite3, json, dbm. default: dbm)
–doit-verbosity	`0` capture (do not print) stdout/stderr from task. `1` capture stdout only. `2` do not capture anything (print everything immediately). Default: 1
–doit-reporter	Where should doit report the output from each task. One of (json, console, zero, executed-only). Default: console
–doit-process	Number of subprocesses (default is 0, i.e. serial processing)
–doit-parallel-type	Tasks can be executed in parallel in different ways: `process`: uses python multiprocessing module `thread`: uses threads. Default is process.

radial_pos

The `gamtools radial_pos` tool is used to calculate chromatin radial positioning from GAM segregation tables. Radial positioning is estimated from the average size of NPs that contain a given chromatin region, since chromatin that occupies a more peripheral position will be intersected by smaller, more apical NPs (i.e. those which slice the nucleus close to the top/bottom/sides), whereas more central chromatin can only be intersected by larger, more equatorial NPs (i.e. those which slice the nucleus through the middle).

The size of each NP is estimated from it's genomic coverage, i.e. the number of positive windows. NPs which contain a larger number of positive windows are assumed to also be larger in volume.

Usage and option summary

Usage:

```
gamtools radial_pos [OPTIONS] -s <SEGREGATION_FILE> -o <OUTPUT_FILE>
```

Optional parameters:

Option	Description
-n, –no-blanks	Exclude regions that were never detected from the output (for making bedgraphs)

select

The `gamtools select` tool is used to select or exclude samples from a segregation table.

Usage and option summary

Usage:

```
gamtools select [OPTIONS] -s <SEGREGATION_FILE> -o <OUTPUT_FILE>
                          -n [<SAMPLE_NAME> [<SAMPLE_NAME> ...]]
```

Required parameters:

Option	Description
-s, –segregation-file	A file containing the segregation of all samples
-n, –sample-names	Names of the samples to remove
-o, –output-file	Output file path (or - to write to stdout)

Optional parameters:

Option	Description
-d, –drop-samples	Discard the listed samples (default: discard samples not in the list)

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

License

GAMtools is freely available under the Apache License, Version 2.0