gat Documentation

Release 1.0

Andreas Heger

Aug 02, 2017

Contents

Welcome to the home page of the Genomic Association Tester (*GAT*).

CHAPTER 1

Overview

A common question in genomic analysis is whether two sets of genomic intervals overlap significantly. This question arises, for example, in the interpretation of ChIP-Seq or RNA-Seq data. Because of complex genome organization, its answer is non-trivial.

The Genomic Association Tester (GAT) is a tool for computing the significance of overlap between multiple sets of genomic intervals. GAT estimates significance based on simulation and can take into account genome organization like isochores and correct for regions of low mapability.

GAT accepts as input standard genomic file formats and can be used in large scale analyses, comparing the association of multiple sets of genomic intervals simultaneously. Multiple testing is controlled using the false discovery rate.

In this manual, the *Introduction* covers the basic concepts of GAT. In order to get an idea of typical use cases, see the *Tutorials* section. The *Usage instructions* section contains a complete usage reference.

CHAPTER 2

Contents

Introduction

A common question in genomic analysis is whether two sets of genomic intervals overlap significantly. This question arises, for example, in the interpretation of ChIP-Seq or RNA-Seq data.

The Genomic Association Tester (GAT) is a tool for computing the significance of overlap between multiple sets of genomic intervals. GAT estimates significance based on simulation.

This introduction covers the *Method basics* and describes the *Sampling method*. It introduces the concept of the *Effective Genome* and explains how to account for *G+C bias*

Method basics

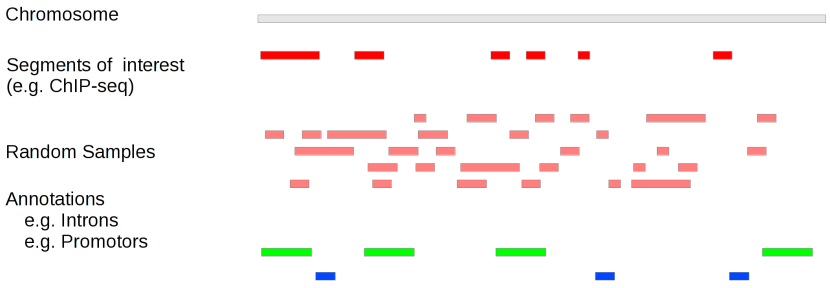
Gat implemements a sampling algorithm. Given a chromosome (*workspace*) and *segments of interest*, for example from a ChIP-Seq experiment, gat creates randomized version of the segments of interest falling into the *workspace*. These *sampled segments* are then compared to existing genomic *annotations*.

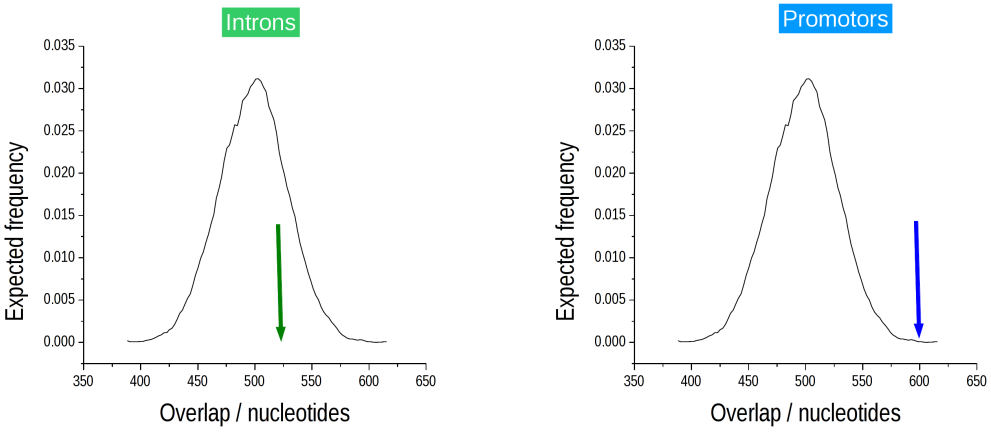
The example below introduces the three sets of genomic intervals. The *workspace*, in this case a single chromosome, is grey. The *segments of interest* and *sampled segments* derived from it are red and light red, (*workspace*) and *segments of interest*, respectively. In this analysis, the *annotations* are the location of introns (green) and promotors (blue) and we use gat to test if the intervals in the ChIP-Seq experiment are enriched in promotors and/or introns.

In the example above, five sets of *sampled segments* have been created. Usually, the number of samples would be much larger (>1000).

Based on the *sampled segments*, the observed overlap is contrasted with the expected overlap and an empirical p-value is determined. The *sampled segments* provide an expectation of the overlap between the segments of interest with a particular annotation. The distribution of the overlap is computed and the empirical p-value is defined as the number of samples that show an equal or greater overlap than the observed overlap.

In the previous example, the overlap with promotors is significant (right), while the overlap with introns (left) is not significantly different from the expectation. We would thus conclude, that our ChIP-Seq intervals are significantly enriched in promotors, but not enriched in introns. Testing for depletion works similarly.


Chromosome
Segments of interest
(e.g. ChIP-seq)
Random Samples
Annotations
e.g. Introns
e.g. Promotors



Sampling method

The sampling method is conceptually simple. Randomized samples of the segments of interest are created in a two-step procedure.

Firstly, a segment size is selected from to same size distribution as the original segments of interest. Secondly, a random position is assigned to the segment. The sampling stops when exactly the same number of nucleotides have been sampled.

To improve the speed of sampling, segment overlap is not resolved until the very end of the sampling procedure. Conflicts are then resolved by randomly removing and re-sampling segments until a covering set has been achieved.

Because the size of randomized segments is derived from the observed segment size distribution of the segments of interest, the actual segment sizes in the *sampled segments* are usually not exactly identical to the ones in the *segments of interest*. This is in contrast to a sampling method that permutes segment positions within the workspace.

Effective genome

Not all regions of the genome are equally accessible to the segments of interest. For example, reads from a ChIP-Seq experiment will never be mapped to a region that is an assembly gap. Failing to account for inaccessible regions causes inflated estimates of fold change and statistical significance.

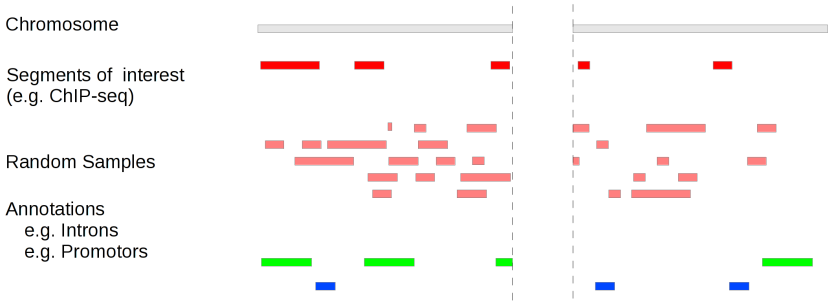
To illustrate this, assume that a particular set of *segments of interest* can only be present on chrX. If we included all chromosomes in the *workspace*, any overlap between the *segments of interest* and any *annotations* on chrX would be overestimated as the *sampled segments* are spread over both chrX and the autosomes.

Such a chromosomal bias is strong and obvious. Often the bias is much more distributed. For example, when comparing orthologous positions between two genomes in a comparative analysis, it is important that the workspace is restricted to only those regions where there is genomic alignment, as orthologous positions will not be found outside of regions that can be aligned.

In order to account for the effective genome, GAT limits the accessible space for simulated segments to a *workspace*, which can be restricted to exclude all regions not appropriate for the analysis. Randomly sampled segments will not fall outside the *workspace*:

What regions need to be excluded from the analysis depends very much on the test being performed. Commonly excluded are for example assembly gaps and regions of low mapability for NGS experiments.

The definition of a *workspace* is crucial and often several *workspace* restrictions need to be combined. For example, in order to test if human long non-coding RNA that are shared with mouse overlap with certain chromatin marks, the


Chromosome
Segments of interest
(e.g. ChIP-seq)
Random Samples
Annotations
e.g. Introns
e.g. Promotors

workspace should be restricted to only those genomic regions in human that align in mouse and also be limited to regions of high mapability.

G+C bias

The human genome (together with many others) is not uniform. Best known is its division into regions of low and high G+C content (*isochores*). Other genomic properties correlate with these, such as gene density, substitution rates, etc. Plus, next-generation sequencing methods display their own G+C biases.

GAT can control for these biases by splitting the genomic region accessible for simulated segments (*workspace*) into smaller regions (*isochores*). Segments are sampled on a per-isochore basis, thus preserving any confounding effects due to different G+C content, before the overall enrichment is computed by combining results from all isochores.

In the example above, G+C content correlates with the density of segments of interest. Regions of low G+C (orange) contain fewer segments than regions of high G+C (purple). Sampling within separate isochores preserves the difference in density.

The *workspace* can be divided into an arbitrary amount of different *isochores*. This is a general technique that can be used to control for different types of bias.

Installation

Requirements

GAT has been written in python and has been tested with the following python versions:

- python 2.7.3
- python 2.6.8

GAT requires the following modules to be installed at installation:

- numpy 1.4 or greater
- cython 0.14 or greater

The plotting and unit test modules also require scipy and matplotlib.

Installing from PyPi

GAT is available at the python package index and can be installed using pip or setuptools.

To install via pip, type:

```
pip install gat
```

To install on OS X, we suggest to begin by installing homebrew by following these instructions

Follow then by:

```
brew install python --with-brewed-openssl
pip install numpy
pip install cython
pip install gat
```


Genome
Isochore workspace
Low G+C
High G+C
Segments of interest
Random samples

Installing via source

The latest changes can be obtained by cloning the repository on **github_**:

```
git clone https://github.com/AndreasHeger/gat.git
```

To install, type:

```
python setup.py install
```

in the package directory.

Release History

1.2.2 Minor features

- Added –random-seed as option.
- moved documentation to read-the-docs.

1.2.1 Bugfix release:

- added missing files `requires.txt` to tarball

1.2 Bugfix release:

- Command line options renamed for CGAT compatibility
- minor bugfixes

1.1 Bugfix release: easier Galaxy integration

- Changed to distutils (from distribute)
- Changed /bin/env to /usr/bin/env

1.0 Release coinciding with publication

0.2

- First release

0.1

- Alpha release

Tutorials

Please see below a collection of tutorials that introduce gat and how it can be used to answer a variety of questions in computational genomics.

Tutorial - Interval overlap

This tutorial demonstrates the usage of *gat* with a simple example - do the binding sites of a transcription factor overlap with DNAse hypersensitive sites?

This tutorial uses the SRF data set described in Valouev et al. (2008). The data sets used in this tutorial are available at:

http://www.cgat.org/~andreas/documentation/gat-examples/TutorialIntervalOverlap.tar.gz

The data is in `srf.hg19.bed`. This *bed* formatted file contains 556 high confidence peaks from the analysis of Valouev et al. (2008) mapped to human chromosome hg19.

This tutorial concentrates on obtaining the data required for a GAT analysis.

First analysis

gat requires three sets of intervals:

1. a set of segments delineating the active part of the genome (*workspace*), and
2. a set of segments of interest (*tracks*), and
3. a set of segments with *annotations*.

gat accepts *bed* formatted files as input.

As segments of interest we will be using the `srf.hgf19.bed` containing the results of the ChIP-Seq experiment:

```
chr5    60627981        60628031        SRF.1
chr5    137801055       137801105       SRF.2
chr5    137800766       137800816       SRF.3
chr7    5570273 5570323 SRF.4
chr5    137827838       137827888       SRF.5
...
```

As our *workspace* we will for now use the *bed* formatted `contigs.bed`, which simply lists all chromosomes in hg19:

```
chr13   0       115169878       ws
chr12   0       133851895       ws
chr11   0       135006516       ws
chr10   0       135534747       ws
chr17   0       81195210        ws
...
```

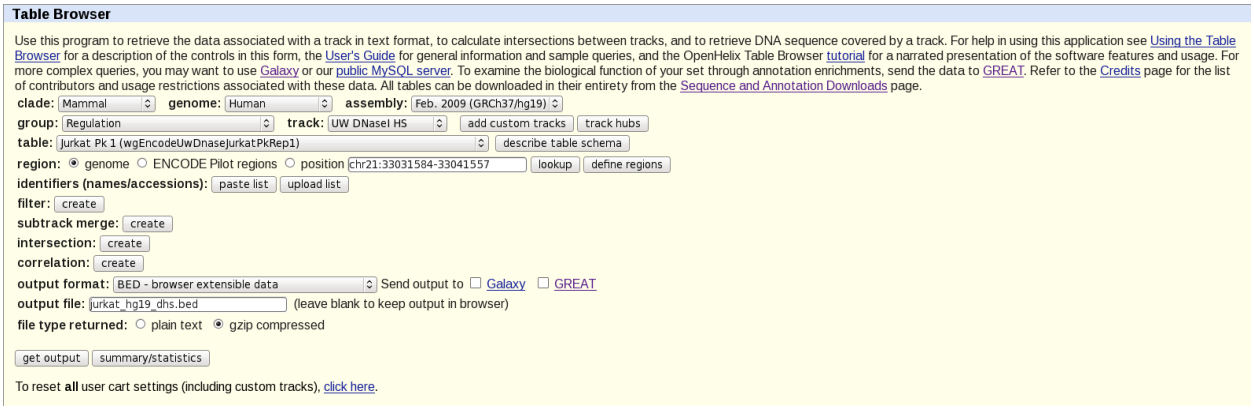
The question we ask is whether within the genome, SRF binding events are in regions of open chromatin as identified by DNAse I hypersensitive sites.

There are many sources for *bed* files. Not having the data ourselves, we make use of data deposited by the ENCODE project within the UCSC genome browser.

To find the relevant data, we start by searching for the term `Jurkat` with `track search`. We find the page with a list of ENCODE datasets:

http://genome.ucsc.edu/cgi-bin/hgTrackUi?g=wgEncodeUwDnase&hgsid=337398699

Selecting `Jurkat`, we can then open the `table browser` and download *bed* formatted coordinates directly:



Alternative ways to obtain and manipulate bed-files are galaxy and bedtools.

We can now run *gat* by giving specifying the three input files:

```
gat-run.py
   --segments=srf.hg19.bed.gz
   --annotations=jurkat.hg19.dhs.bed.gz
   --workspace=contigs.bed.gz
   --ignore-segment-tracks
   --num-samples=1000 --log=gat.log > gat.tsv
```

The option *–ignore-segment-tracks* tells *gat* to ignore the fourth column in the *tracks* file and assume that all intervals in this file belong to the same *track*. If not given, each interval would be treated separately.

The above statement finishes in a few seconds. With large interval collections or many annotations, *gat* might take a while. It is thus good practice to always save the output in a file. The option *–log* tells gat to save information or warning messages into a separate log file.

The first 10 columns of the output file are the most informative:

track	annotation	observed	expected	CI95low	CI95high	stddev	fold	l2fold	pvalue
merged	tb_wgEncodeUwDnaseJurkatPkRep1	20183	246.5650	96.0000	444.0000	105.5933	81.5301	6.3493	1.0000e-03

The table states that we observe an overlap of 20,183 nucleotides, but would expect an overlap of 247 nucleotides, which is an 82 fold enrichment. This is highly significant (p-value of 0.001).

Note that the number of simulations determines the minimum P-value that can be reported. Here, we did 1,000 simulations, thus the minimum P-value we can obtain is 0.001. Usually a few simulations (100) are required to get a good idea about enrichment. For publication, more simulations are required (>10,000) to get a good idea of the statistical significance.

More samples increases memory and runtime requirements of GAT. The computation above took 11 seconds on our local system. Increasing the number of simulations to 10.000 increases the runtime to 103 seconds. Note how the runtime increases linearly with the number of samples.

Do the results make sense? Instead of the Jurkat cells, we can test for enrichment with DHS sites in hepatocytes.

We obtain a *bed*-file as before from UCSC and ENCODE (cell line `hepg2`) and save it as `hepg2.hg19.dhs.bed.gz`. Next, we run GAT with this file instead:

```
gat-run.py --segments=srf.hg19.bed.gz
           --annotations=hepg2.hg19.dhs.bed.gz
           --workspace=contigs.bed.gz
           --ignore-segment-tracks
```

```
            --num-samples=1000
            --log=gat-hepg-unique.tsv.log
```

GAT reports:

track	annotation	observed	expected	CI95low	CI95high	stddev	fold	l2fold	pvalue
merged	tb_wgEncodeUwDnaseHepg2HotspotsRep5	18063	597.1380	339.0000	883.0000	166.9945	31.7084	4.9868	1.0000e-03

Note how the fold enrichment is now less (32 fold), though still highly significant. This is the expected result, DHS sensitive sites are shared among different tissue types.

We can test this by comparing the two different DHS sets against each other:

```
gat-run.py --segments=hepg2.hg19.dhs.bed.gz
           --annotations=jurkat.hg19.dhs.bed.gz
           --workspace=contigs.bed.gz --ignore-segment-tracks --num-samples=1000 >␣
↪dhs.tsv
```

track	annotation	observed	expected	CI95low	CI95high	stddev	fold	l2fold	pvalue	qvalue
merged	tb_wgEncodeUwDnaseJurkatRep5	6168	928.2774	403565.0000	470129.0000	19.7800	3.4890	3.7537	1.0000e-03	1.0000e-03

Indeed, the overlap between DHS sites is significant (pvalue = 0.001). We observe a 6.2Mb overlap, but expect only a 0.5Mb overlap. This is a 13-fold enrichment.

The runtime has increased from 11s to 308s. Apart from the number of samples, the number of segments in the *segments of interest* are a major determinant of the time it takes to complete a run. More information about memory and time requirement of GAT are in the section about GAT *Performance*.

Let us try removing all intervals from `hepg2.hg19.dhs.bed.gz` that overlap sites found in Jurkat cells using bedtools:

```
intersectBed -a hepg2.hg19.dhs.bed.gz -b jurkat.hg19.dhs.bed.gz -wa -v | gzip  >␣
↪hepg2_unique.dhs.bed.gz
```

106,308 segments out of 144,172 remain.

Next, we re-run the GAT analysis:

```
gat-run.py --segments=srf.hg19.bed.gz --annotations=hepg2-unique.hg19.dhs.bed.gz --
↪workspace=contigs.bed.gz  \
           --ignore-segment-tracks      \
            --num-samples=1000 --log=gat-hepg-unique.tsv.log
```

track	annotation	observed	expected	CI95low	CI95high	stddev	fold	l2fold	pvalue	qvalue
merged	.	425	324.6790	143.0000	539.0000	117.8233	1.3080	0.3874	1.8500e-01	1.8500e-01

Now, we observe only a 30% enrichment and this is not significant (P-value 0.19). The observed overlap of 425 nucleotides is very close to the expected 325 nucleotides. We conclude, that the overlap of SRF between DHS sites in hepatocytes is due to those DHS sites that are shared between hepatocytes and Jurkat cells.

This example showed how GAT can be used in a very simple scenario to test if two genomic features are associated with each other. The following tutorials will introduce more complex usage, for example using the effective genome and testing multiple annotations simultaneously.

Tutorial - Genomic annotation

This tutorial demonstrates the usage of *gat* with a simple example - where does a transcription factor bind in the genome?

As opposed to *Tutorial - Interval overlap* we will not be looking at the overlap between one set of intervals with another, but at the overlap of one set of intervals with multiple others.

This tutorial uses the SRF data set described in Valouev et al. (2008). The data sets used in this tutorial are available at:

http://www.cgat.org/~andreas/documentation/gat-examples/TutorialGenomicAnnotation.tar.gz

The data is in `srf.hg19.bed.gz`. This *bed* formatted file contains 556 high confidence peaks from the analysis of Valouev et al. (2008) mapped to human chromosome hg19.

We build the analysis in multiple steps. First, we will perform a simple analysis, which will also motivate the use of *gat*. Later, we will build more sophisticated analyses that take into account the effective genome.

First analysis

gat requires three sets of intervals:

1. a set of segments delineating the active part of the genome (*workspace*), and
2. a set of segments of interest (*tracks*), and
3. a set of segments with *annotations*.

gat accepts *bed* formatted files as input.

As segments of interest we will be using the `srf.hgf19.bed.gz` containing the results of the ChIP-Seq experiment:

```
chr5    60627981        60628031        SRF.1
chr5    137801055       137801105       SRF.2
chr5    137800766       137800816       SRF.3
chr7    5570273 5570323 SRF.4
chr5    137827838       137827888       SRF.5
...
```

As our *workspace* we will for now use the *bed* formatted `contigs.bed`, which simply lists all chromosomes in hg19:

```
chr13   0       115169878       ws
chr12   0       133851895       ws
chr11   0       135006516       ws
chr10   0       135534747       ws
chr17   0       81195210        ws
...
```

As our annotations file, we will use the `annotations_geneset.bed.gz`.

This file required a little more effort to build. We took all protein genes of Ensembl (release 67) and merged the exons of all transcripts of a gene. Based on these gene definitions we then divided genomic regions into intergenic, intronic and exonic regions. We also annotated the UTR (5' and 3'), and the 5kb flank upstream and downstream. The result is a set of non-overlapping intervals covering the full genome:

```
...
chr1    362640  367639  5flank
chr1    367640  367658  UTR5
chr1    367659  368594  CDS
chr1    368595  368634  UTR3
chr1    368635  373634  3flank
chr1    373635  616058  intergenic
chr1    616059  621058  3flank
chr1    621059  621098  UTR3
chr1    621099  622034  CDS
chr1    622035  622053  UTR5
chr1    622054  627053  5flank
...
```

We can now run *gat* by giving specifying the three input files:

```
gat-run.py --ignore-segment-tracks --segments=srf.hg19.bed.gz
   --annotations=annotations_geneset.bed.gz --workspace=contigs.bed.gz
--num-samples=1000 --log=gat.log > gat.out
```

The option *–ignore-segment-tracks* tells *gat* to ignore the fourth column in the *tracks* file and assume that all intervals in this file belong to the same *track*. If not given, each interval would be treated separately.

The above statement finishes in a few seconds. With large interval collections or many annotations, *gat* might take a while. It is thus good practice to always save the output in a file. The option *–log* tells gat to save information or warning messages into a separate log file.

The first 11 columns of the output file are the most informative:

track	annotation	observed	expected	Ci95low	CI95high	stddev	fold	l2fold	pvalue	qvalue
merged	intergenic	5800	14056.330	13100.000	15000.000	583.7181	0.4126	-1.2771	1.0000e-03	1.5714e-03
merged	intronic	8816	10633.853	09665.0000	11602.000	0592.7589	0.8291	-0.2705	1.0000e-03	1.5714e-03
merged	UTR3	233	278.0720	100.0000	493.0000	117.3112	0.8379	-0.2551	3.6500e-01	4.4611e-01
merged	3flank	800	659.6560	400.0000	1000.0000	175.0544	1.2128	0.2783	2.3100e-01	3.1762e-01
merged	CDS	754	360.7680	161.0000	580.0000	127.2204	2.0900	1.0635	1.0000e-03	1.5714e-03
merged	flank	1334	167.8620	50.0000	350.0000	91.4581	7.9470	2.9904	1.0000e-03	1.5714e-03
merged	5flank	6524	691.5400	400.0000	1000.0000	185.0053	9.4340	3.2379	1.0000e-03	1.5714e-03
merged	UTR5	3441	87.0110	0.0000	200.0000	60.9119	39.5467	5.3055	1.0000e-03	1.5714e-03

The first two columns contain the name of the *track* and *annotation* that are being compared. The columns *observed* and *expected* give the observed and expected nucleotide overlap, respectively, between the *track* and *annotation*.

The following columns CI95low, CI95high, stddev give 95% confidence intervals and the standard deviation of the sample distribution, respectively.

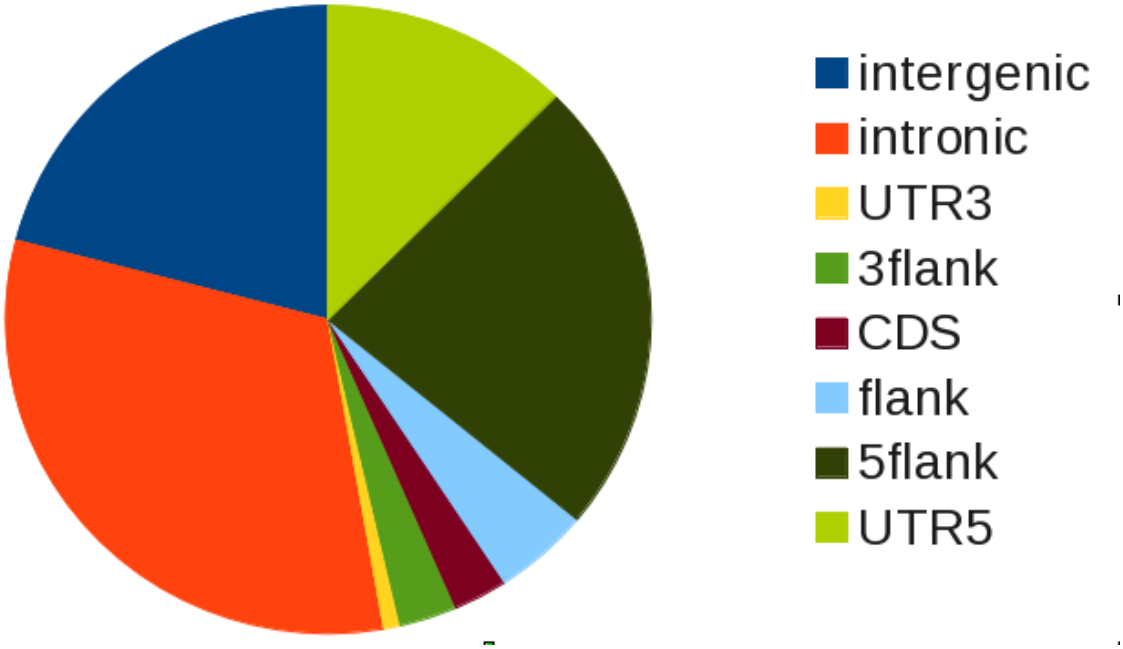
The *fold* column is the fold enrichment or depletion and is computed as the ratio of *observed* over *expected*. The column *l2fold* is the log2 of this ratio.

The column *pvalue* gives the empirical *p-value*, i.e. in what proportion of samples was a higher enrichment or lower

depletion found than the one that was observed.

The column *qvalue* lists a multiple testing corrected *p-value*. Setting a qvalue threshold and accepting only those comparisons with a qvalue below that threshold corresponds to controlling the false discovery rate at that particular level.

What does this table tell us? Looking at the column *observed* only, we see that most binding of SRF occurs in intronic and intergenic regions:



Strictly speaking, this is a a naive analysis that does not require *gat*. The observed overlap alone does not tell us if the overlap we see is more or less than we expect. We do know that there are much more and larger intronic regions than there are UTRs, for example.

More instructive is to look at the enrichment within the various genomic regions, which is given by the *fold* change.

Here, we clearly see that SRF binds preferentially at transcription start sites (UTR5 and 5flank), while its binding is actually less than expected in introns and intergenic regions.

The effective genome

In the previous analysis we used the complete genome (3.1Gb) as the *workspace*. However, that is not realistic. For example, SRF will not be predicted in regions that are assembly gaps. Generally speaking, if the workspace is too large, fold enrichment values will be too optimistic.

To get a more accurate estimate of the enrichment in various regions, we should exclude assembly gaps.

The *bed* formatted file `contigs_ungapped.bed.gz` contains only those genomic regions that are not assembly gaps (2.86Gb). We can use this file instead:

```
gat-run.py --ignore-segment-tracks --segments=srf.hg19.bed.gz
   --annotations=annotations_geneset.bed.gz --workspace=contigs_ungapped.bed.gz
   --num-samples=1000 --log=gat.log > gat.out
```

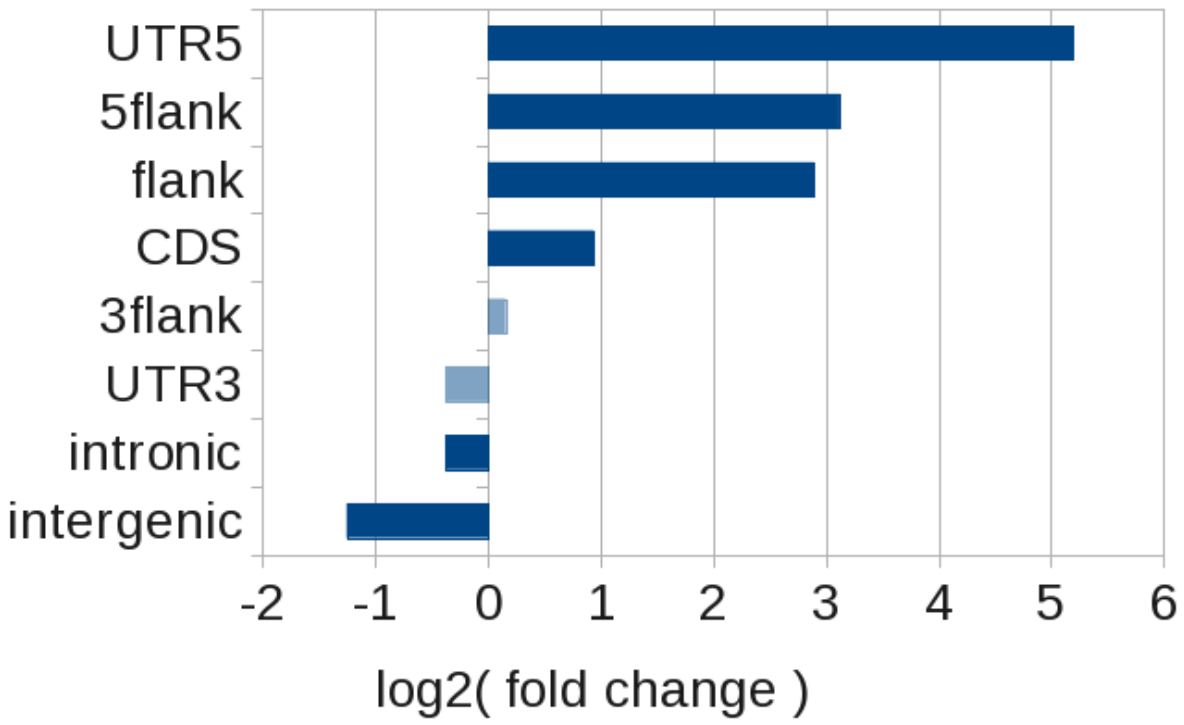


Fig. 2.1: Binding distribution of SRF with respect to known protein coding genes. Plotted is the log2(fold change). Value not significant are transparent.

annotation	observed	expected	fold	l2fold	pvalue	qvalue
intergenic	5800	13806.4540	0.4201	-1.2512	1.0000e-03	2.2000e-03
UTR3	233	303.6340	0.7674	-0.3820	2.5300e-01	3.9757e-01
intronic	8816	11473.2200	0.7684	-0.3801	1.0000e-03	2.2000e-03
3flank	800	713.4290	1.1213	0.1652	3.4000e-01	4.6750e-01
CDS	754	391.1840	1.9275	0.9467	5.0000e-03	9.1667e-03
flank	1334	182.0200	7.3289	2.8736	1.0000e-03	2.2000e-03
5flank	6524	761.1600	8.5711	3.0995	1.0000e-03	2.2000e-03
UTR5	3441	97.3670	35.3405	5.1433	1.0000e-03	2.2000e-03

The associated fold changes change, albeit not much. But have we done enough? The SRF intervals are the result of a ChIP-Seq experiment. Because these were short reads (25bp), not all can be unambiguously mapped to a unique genomic location. This again effectively removes some genomic regions from the analysis.

The *bed* formatted `mapability_36.filtered.bed.gz` contains all those genomic regions, that are uniquely mapable with reads of 24 bases. These regions have been derived from the UCSC mapability tracks and reduce the effective genome considerably (1.96Gb).

We could intersect the two bed files ourselves, but we can also supply multiple workspaces to *gat*. *gat* will automatically intersect multiple workspaces:

```
gat-run.py --ignore-segment-tracks --segments=srf.hg19.bed.gz
   --annotations=annotations_geneset.bed.gz
   --workspace=contigs_ungapped.bed.gz
   --workspace=mapability_36.filtered.bed.gz
   --num-samples=1000 --log=gat.log > gat.out
```

As a consequence of reducing the workspace the fold changes change:

annotation	observed	expected	fold	l2fold	pvalue	qvalue
intergenic	5800	12531.2490	0.4628	-1.1114	1.0000e-03	1.6000e-03
UTR3	233	385.1620	0.6049	-0.7251	1.1000e-01	1.2571e-01
intronic	8816	10942.7440	0.8056	-0.3118	1.0000e-03	1.6000e-03
3flank	800	625.3780	1.2792	0.3553	1.6500e-01	1.6500e-01
CDS	754	540.3700	1.3953	0.4806	8.2000e-02	1.0933e-01
flank	1334	166.6400	8.0053	3.0010	1.0000e-03	1.6000e-03
5flank	6524	638.2110	10.2223	3.3537	1.0000e-03	1.6000e-03
UTR5	3441	122.2010	28.1585	4.8155	1.0000e-03	1.6000e-03

Tutorial - Functional annnotation

This tutorial demonstrates the use of *gat* for functional annotation. The tutorial follows the analysis in the MacLean et al. (2010) paper introducing GREAT.

The data sets used in this tutorial are available at:

http://www.cgat.org/~andreas/documentation/gat-examples/TutorialFunctionalAnnotation.tar.gz

We are interested if the binding sites of SRF predicted from our ChIP-Seq experiment are enriched in the regulatory domains of genes of specific functions.

The data is in `srf.hg19.bed`. This *bed* formatted file contains 556 high confidence peaks from the analysis of Valouev et al. (2008) mapped to human chromosome hg19.

Functional annotation with GREAT

gat comes with a tool called `gat-great` that computes enrichment statistics using the binomial test implemented in GREAT.

To do an analysis as implemented in GREAT, we have prepared a *bed* formatted file (`regulatory_domains.bed`) with regulatory domains using GREAT's basal+extension rule.

In GREAT's definition, the regulatory domain of a gene contains a basal region and an optional extension. The basal region is defined as the region 5kb upstream and 1kb downstream of the transcription start site of a representative transcript. The basal region is then extended up to 1Mb in either direction but only up to the basal region of the closest domains.

In this example, we have used the transcription start sites of the ENSEMBL human protein coding gene set of Ensembl (release 67).

Each gene was replaced with GO terms associated with the gene obtained from the GO Gene Ontology definitions. GO terms were mapped via uniprot identifiers to genes and ancestral ontology terms were inferred.

GREAT excludes assembly gaps in the genome from the analysis. The *bed* formatted `contigs_ungapped.bed` contains all genomic regions exclusive of assembyl gaps.

We have now all data in place to run a GREAT analysis:

```
gat-great.py --verbose=5 \
             --log=great.log \
             --segments=srf.hg19.bed \
             --annotations=regulatory_domains.bed \
             --workspace=contigs_ungapped.bed \
             --ignore-segment-tracks \
             --qvalue-method=BH \
             --descriptions=go2description.tsv \
>& great.tsv
```

We also added a file `go2description.tsv` that contains a table with descriptions for GO identifiers.

We inserted the table into an SQL database for easy analysis. These are the top 10 results:

annotation	pvalue	observed	expected	description
GO:0015629	1.84357379006e-11	52	18.0074449104	“actin cytoskeleton”
GO:0032796	4.22734641877e-09	5	0.0557976925372	“uropod organization”
GO:0070688	1.05832288791e-07	7	0.358045266782	“MLL5-L complex”
GO:0043229	2.56771180099e-07	424	369.125027893	“intracellular organelle”
GO:0044424	3.38999777659e-07	457	406.64284565	“intracellular part”
GO:0043226	3.77332356747e-07	424	369.977837368	organelle
GO:0072301	1.28008163954e-06	3	0.0198635059219	“positive regulation of glomerular metanephric mesangial cell proliferation”
GO:0001725	1.96123184157e-06	12	2.08962341062	“stress fiber”
GO:0005634	2.30813240931e-06	295	240.687702571	nucleus
GO:0045214	2.78146278125e-06	11	1.79113154372	“sarcomere organization”

Validation against GREAT

We can check if the results are comparable to the GREAT server. We submitted our segments to GREAT, downloaded all the results into `srf.great.all.tsv` and loaded them into an SQL database. These are the top 10 results:

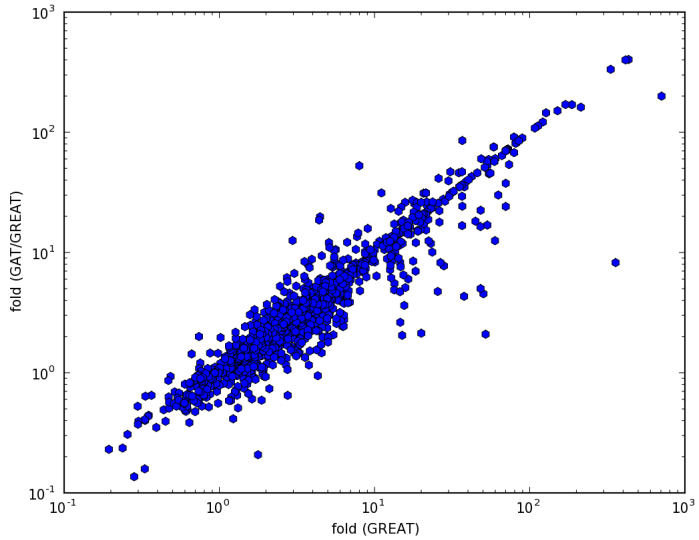
ID	BinomP	ObsRegions	ExpRegions	Desc
GO:0015629	3.064707e-11	51	17.68622	“actin cytoskeleton”
GO:0032796	4.223825e-09	5	0.05578831	“uropod organization”
GO:0045214	1.057722e-08	10	0.7800466	“sarcomere organization”
GO:0030863	2.205659e-08	16	2.666392	“cortical cytoskeleton”
GO:0001725	1.786459e-07	13	1.991518	“stress fiber”
GO:0044424	2.007378e-07	479	431.1647	“intracellular part”
GO:0070688	2.151319e-07	7	0.3981916	“MLL5-L complex”
GO:0005634	2.556219e-07	311	251.391	nucleus
GO:0032432	2.931657e-07	13	2.081847	“actin filament bundle”
GO:0043229	3.557439e-07	443	390.9065	“intracellular organelle”

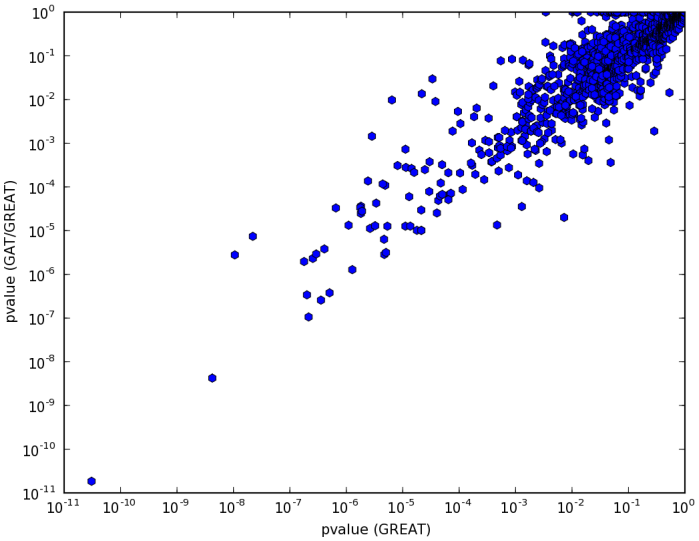
The top 10 results are comparable. The same holds generally for fold change values:

and pvalues:

Some differences are to be expected:

1. We use the ENSEMBL gene set, while GREAT uses Refseq.
2. We use a different definition of representative transcripts.
3. Our coordinates of alignment gaps might differ.
4. The assignment of GO terms to genes differ.
5. The implementations might differ in some details.


fold (GAT/GREAT)
fold (GREAT)

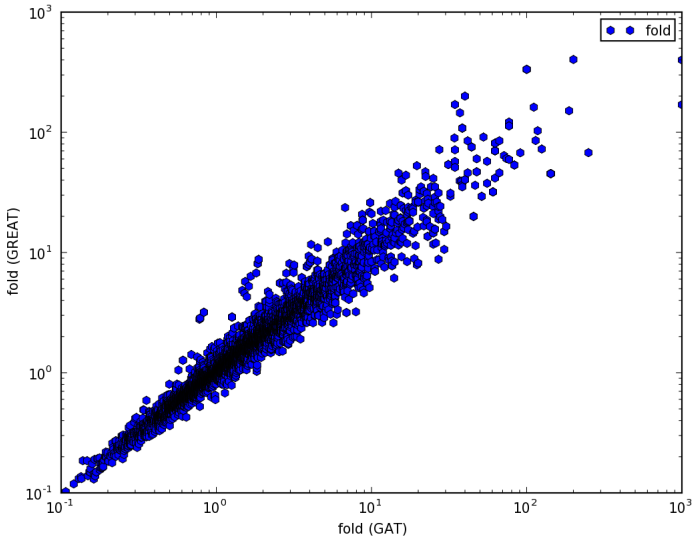

pvalue (GAT/GREAT)
pvalue (GREAT)

Functional annotation with gat

Gat can be run with the same input as we used for great:

```
gat-run.py --verbose=5 \
           --log=gatnormed.tsv.log \
            --segments=srf.hg19.bed \
            --annotations=regulatory_domains.bed \
            --workspace=contigs_ungapped.bed \
            --ignore-segment-tracks \
            --qvalue-method=BH \
            --descriptions=go2description.tsv \
            --pvalue-method=norm \
            >& gatnormed.tsv
```

Fold changes are very similar:



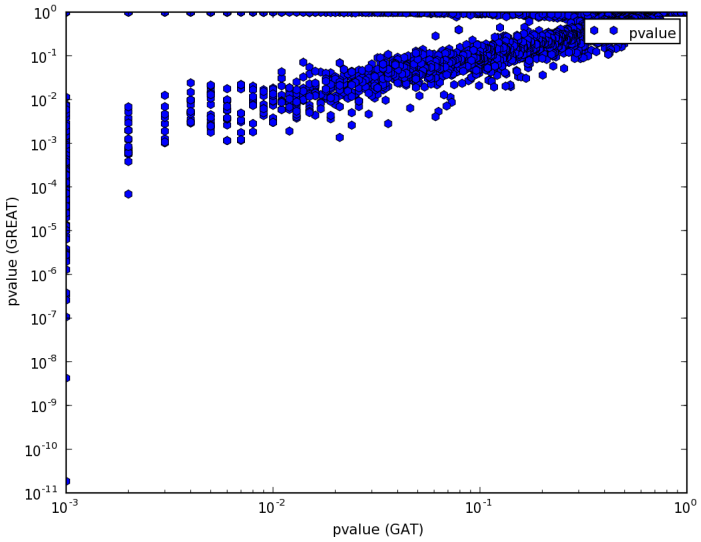
but the p-value comparison shows interesting pattern:

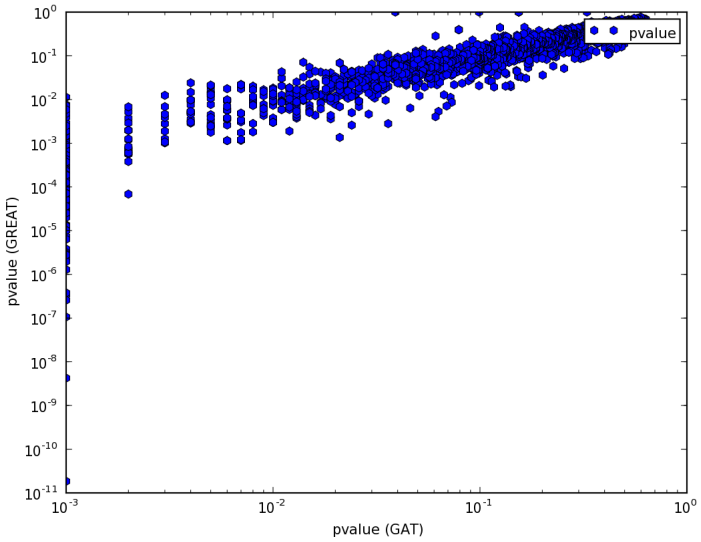
The pattern is explained easily. GREAT computes only the P-Value for enrichment, while GAT computes P-Value both for enrichment and depletion. Indeed, if we only plot p-values for annotations that are enriched, the values are comparable:

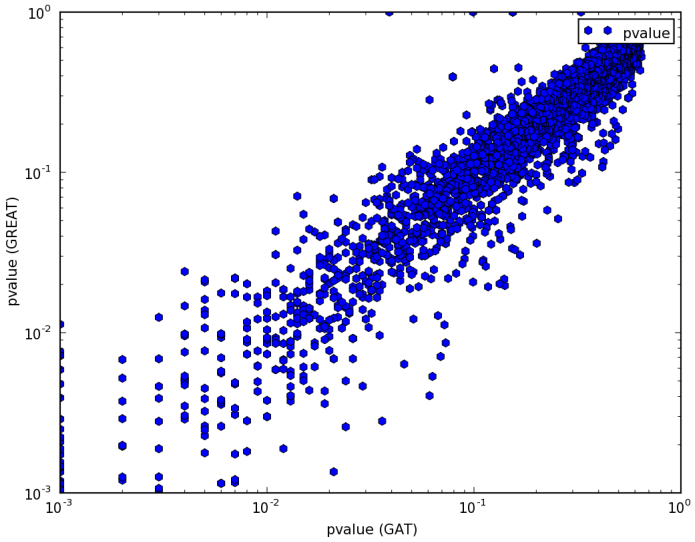
Note how the p-values are very well correlated above 10E-3:

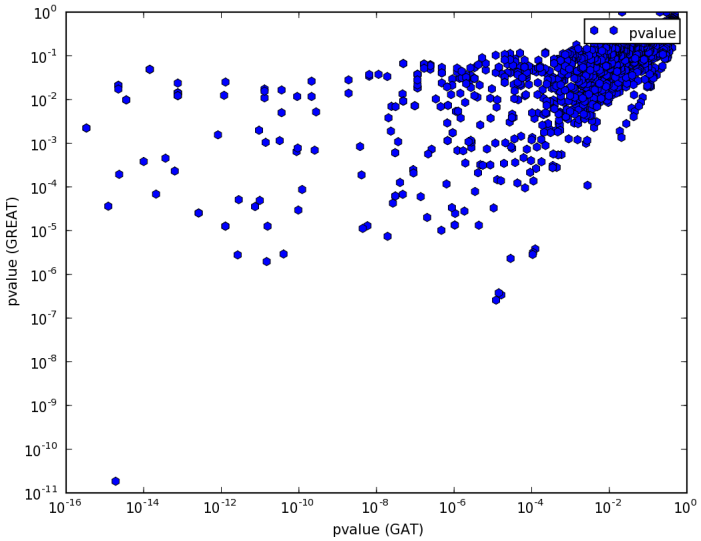
Below a p-Value of 10E-3 the correlation breaks down. Unfortunately, the lowest empirical p-value is determined by the number of simulations performed. Adding more simulations will allow us to estimate lower p-values, but will also increase the runtime. A short-cut is to extrapolate from lower p-values by adding the option `--pvalue-method=norm`:

The table with enriched categories is dominated by small categories with very little expected overlap leading to very large fold changes:


pvalue
pvalue (GREAT)
pvalue (GAT)


pvalue
pvalue (GREAT)
pvalue (GAT)


pvalue
pvalue (GREAT)
pvalue (GAT)
10^{0}
10^{-1}
10^{-2}
10^{-3}


pvalue
pvalue (GREAT)
pvalue (GAT)
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-16}
10^{-14}
10^{-12}
10^{-10}
10^{-8}
10^{-6}
10^{-4}
10^{-2}
10^{0}

annotation	pvalue	observed	expected	fold	description
GO:0043495	0.0	200	10.8	18.5185	“protein anchor”
GO:0070688	0.0	350	16.7	20.9581	“MLL5-L complex”
GO:0000212	0.0	200	9.25	21.6216	“meiotic spindle organization”
GO:0045896	0.0	150	4.65	32.2581	“regulation of transcription during mitosis”
GO:0045897	0.0	150	4.65	32.2581	“positive regulation of transcription during mitosis”
GO:0046022	0.0	150	4.65	32.2581	“positive regulation of transcription from RNA polymerase II promoter during mitosis”
GO:0046021	0.0	150	4.65	32.2581	“regulation of transcription from RNA polymerase II promoter, mitotic”
GO:0071895	0.0	100	2.7	37.037	“odontoblast differentiation”
GO:0032796	0.0	250	6.65	37.594	“uropod organization”
GO:0021593	0.0	100	2.65	37.7358	“rhombomere morphogenesis”

For interpretation of the results it is often advisable to remove annotations that are rare.

annotation	pvalue	observed	expected	fold	description
GO:0015629	3.3307e-16	2600	932.458	2.7883	“actin cytoskeleton”
GO:0044425	5.5445e-11	9620	13468.369	0.7143	“membrane part”
GO:0016021	5.7537e-11	7870	11699.204	0.6727	“integral to membrane”
GO:0031224	1.2393e-10	8220	11999.454	0.685	“intrinsic to membrane”
GO:0016020	5.5517e-08	12970	16059.768	0.8076	membrane
GO:0004888	1.0775e-06	1000	2534.458	0.3946	“transmembrane signaling receptor activity”
GO:0030029	1.1332e-06	2450	1278.537	1.9163	“actin filament-based process”
GO:0030036	1.3077e-06	2300	1174.087	1.959	“actin cytoskeleton organization”
GO:0003779	3.2187e-06	1900	963.501	1.972	“actin binding”
GO:0005886	5.2155e-06	7720	10170.117	0.7591	“plasma membrane”

Currently, we estimate fold enrichment for categories within a workspace that excludes ungapped regions. As before (*Tutorial - Interval overlap*), a thorough analysis should also exclude regions of low mapability.

Usage instructions

This page describes basic and advanced usage of GAT.

A list of all command-line options is available via:

```
gat-run.py --help
```

Basic usage

The *gat* tool is controlled via the `gat-run.py` script. This script requires the following input:

1. A set of *intervals* `S` with *segments of interest* to test.
2. A set of *intervals* `A` with *annotations* to test against.
3. A set of *intervals* `W` describing a *workspace*

GAT requires *bed* formatted files. In its simplest form, GAT is then run as:

```
gat-run.py
   --segment-file=segments.bed.gz
```

```
--workspace-file=workspace.bed.gz
--annotation-file=annotations.bed.gz
```

The script recognizes *gzip* compressed files by the suffix `.gz`.

The principal output is a tab-separated table of pairwise comparisons between each *segments of interest* and *annotations*. The table will be written to stdout, unless the option `--stdout` is given with a filename to which output should be redirected.

The main columns in the table are:

track the *segments of interest track*

annotation the *annotations track*

observed the observed count

expected the expected count based on the *sampled segments*

CI95low the value at the 5% percentile of *samples*

CI95high the value at the 95% percentile of *samples*

stddev the standard deviation of *samples*

fold the fold enrichment, given by the ratio observed / expected

l2fold log2 of the fold enrichment value

pvalue the *p-value* of enrichment/depletion

qvalue a multiple-testing corrected *p-value*. See *multiple testing correction*.

Additionally, there are the following columns:

track_nsegments number of segments in *track* in *segments of interest*

track_size number of residues in covered by *track* in *segments of interest* within the *workspace*

track_density fraction of residues in *track* in *segments of interest* within the *workspace*

annotation_nsegments number of segments in *track* in *annotations*.

annotation_size number of residues in covered by *track* in *annotations* within the *workspace*

annotation_density number of residues in covered by *track* in *annotations* within the *workspace*

overlap_nsegments number of segments in overlapping between *segments of interest* and *annotations*

overlap_size number of nucleotides overlapping between *segments of interest* and *annotations*

overlap_density fraction of residues overlapping between *segments of interest* and *annotations* within *workspace*

percent_overlap_nsegments_track percentage of segments in *segments of interest* overlapping *annotations*

percent_overlap_size_track percentage of nucleotides in *segments of interest* overlapping *annotations*

percent_overlap_nsegments_annotation percentage of segments in *annotations* overlapping *segments of interest*

percent_overlap_size_annotation percentage of nucleotides in *annotations* overlapping *segments of interest*

description additional description of track (requires `--descriptions` to be set).

Further output files such as auxiliary summary statistics go to files named according to `--filename-output-pattern`. The argument to `filename-output-pattern` should contain one `%s` placeholder, which is then substituted with section names.

Count here denotes the measure of association and defaults to *number of overlapping nucleotides*.

Advanced Usage

Submitting multiple files

All of the options *–segment-file*, *–workspace-file*, *–annotation-file* can be used several times on the command line. What happens with multiple files depends on the file type:

1. Multiple *–segment-file* entries are added to the list of *segments of interest* to test with.
2. Multiple *–annotation-file* entries are added to the list of *annotations* to test against.
3. Multiple *–workspace* entries are intersected to create a single workspace.

Generally, *gat* will test *m segments of interest* lists against *n annotations* lists in all *m* * *n* combinations.

Within a *bed* formatted file, different *tracks* can be separated using a UCSC formatted `track` line, such as this:

```
track name="segmentset1"
chr1  23     100
chr3  50     2000
track name="segmentset2"
chr1  1000   2000
chr3  4000   5000
```

or alternatively, using the fourth column in a *bed* formatted file:

```
chr1 23      100     segmentset1
chr3 50      2000    segmentset1
chr1 1000    2000    segmentset2
chr3 4000    5000    segmentset2
```

The latter takes precedence. The option *–ignore-segment-tracks‘* forces gat to ignore the fourth column and consider all intervals to be from a single interval set.

Note: Be careful with bed-files where each interval gets a unique identifier. Gat will interprete each interval as a separate segment set to read. This is usually not intended and causes gat to require a very large amount of memory. (see the option `--ignore-segment-tracks`

By default, tracks can not be split over multiple files. The option `--enable-split-tracks` permits this.

Adding isochores

Isochores are genomic segments with common properties that are potentially correlated with the segments of interest and the annotations, but the correlation is not of interest here. For example, consider a CHiP-Seq experiment and the testing if CHiP-Seq intervals are close to genes. G+C rich regions in the genome are gene rich, while at the same time there is possibly a nucleotide composition bias in the CHiP-Seq protocol depleting A+T rich sequence. An association between genes and CHiP-Seq intervals might simply be due to the G+C effect. Using isochores can control for this effect to some extent.

Isochores split the *workspace* into smaller workspaces of similar properties, so called *isochore workspaces*. Simulations are performed for each *isochore workspaces* separately. At the end, results for each all isochore workspaces are aggregated.

In order to add isochores, use the *–isochore-file* command line option.

Choosing measures of association

Counters describe the measure of association that is tested. Counters are selected with the command line option `--counter`. Available counters are:

1. `nucleotide-overlap`: number of bases overlapping [default]
2. `segment-overlap`: number of intervals intervals in the *segments of interest* overlapping *annotations*. A single base-pair overlap is sufficient.
3. `segment-mid-overlap`: number of intervals in the *segments of interest* overlapping at their midpoint *annotations*.
4. `annotations-overlap`: number of intervals in the *annotations* overlapping *segments of interest*. A single base-pair overlap is sufficient.
5. `segment-mid-overlap`: number of intervals in the *annotations* overlapping at their midpoint *segments of interest*

Multiple counters can be given. If only one counter is provided, the output will be to stdout. Otherwise, separate output files will be created each counter. The filename can be controlled with the `--output-table-pattern` option.

Changing the PValue method

By default, *gat* returns the empirical *p-value* based on the sampling procedure. The minimum *p-value* is `1 / number of samples`.

Sometimes the lower bound on p-values causes methods that estimate the FDR to fail as the distribution of p-values is atypical. In order to estimate lower pvalues, the number of samples needs to be increased. Unfortunately, the run-time of gat is directly proportional to the number of samples.

A solution is to set the option `--pvalue-method` to `--pvalue-method=norm`. In that case, pvalues are estimated by fitting a normal distribution to the samples. Small p-values are obtained by extrapolating from this fit.

Multiple testing correction

gat provides several methods for controlling the false discovery rate. The default is to use the Benjamini-Hochberg procedure. Different methods can be chosen with the `--qvalue-method` option.

`--qvalue-method=storey` uses the procedure by Storey et al. (2002) to compute a *q-value* for each pairwise comparison. The implementation is in its functionality equivalent to the qvalue package implemented in R.

Other options are equivalent to the methods as implemented in the R function `p.adjust`.

Caching sampling results

gat can save and retrieve samples from a cache `--cache=cache_filename`. If `cache_filename` does not exist, samples will be saved to the cache after computation. If `cache_filename` does already exist, samples will be retrieved from the cache instead of being re-computed. Using cached samples is useful when trying different counters (see *Choosing measures of association*).

If the option `--counts-file` is given, *gat* will skip the sampling and counting step completely and read observed counts from `--count-file=counts_filename`.

Using multiple CPU/cores

GAT can make use of several available CPU/cores if available. Use the `--num-threads=#` option in order to specify how many CPU/cores GAT will make use of. The default `--num-threads=0` means that GAT will not use any multiprocessing.

Outputting intermediate results

A variety of options govern the output of intermediate results by gat. These options usually accept patterns that represent filenames with a `%s` as a wild card character. The wild card is replaced with various keys. Note that the amount of data output can be substantial.

`--output-counts-pattern` output counts. One file is created for each counter. Counts output files are required for *gat-compare*.

`--output-plots-pattern` create plots (requires matplotlib). One plot for each annotation is created showing the distribution of expected counts and the observed count. Also, outputs the distribution of p-values and q-values.

`--output-samples-pattern` output *bed* formatted files with individual samples.

Other tools

gat-compare

The gat-compare tool can be used to test if the fold changes found in two or more different gat experiments are significantly different from each other.

This tool requires the output files with counts created using the `--output-counts-pattern` option.

For example, to compare if fold changes are signficantly different between two cell lines, execute:

```
gat-run.py --segments=CD4.bed.gz <...>
--output-counts-pattern=CD4.%s.overlap.counts.tsv.gz
gat-run.py --segments=CD14.bed.gz <...>
--output-counts-pattern=CD14.%s.overlap.counts.tsv.gz

gat-compare.py CD4.nucleotide-overlap.counts.tsv.gz CD14.nucleotide-overlap.counts.
↪tsv.gz
```

gat-plot

Plot gat results.

gat-great

Perform a GREAT analysis:

```
gat-great.py
   --segment-file=segments.bed.gz
   --workspace-file=workspace.bed.gz
   --annotation-file=annotations.bed.gz
```

Interpreting GAT results

Fold change

Gat reports fold changes. Fold change is simply expressed as a ratio of an observed metric compared to the expected value of the metric based on randomizations.

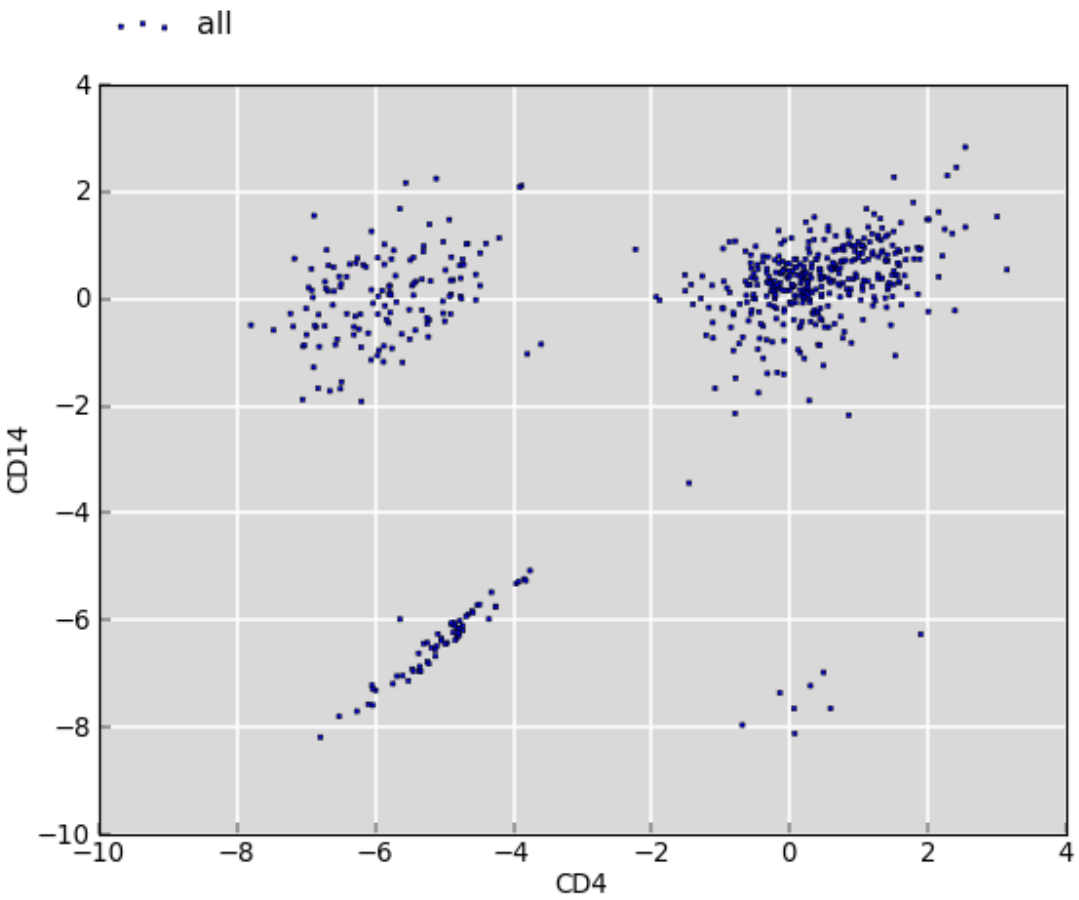
Fold changes of a single set of *segments of interest* against various annotations can be compared directly. Indeed, the primary objective of randomization is to remove the differences between number and segment sizes of different annotations in order to make them comparable. For example, the fold change of a set of transcription factor sites in promotors can directly be contrasted with the fold change of the same sites in introns.

When comparing the fold enrichment of multiple sets of *segments of interest* against the same annotation, some caution must be exercised. Here, the question is to compare the fold change of sites of transcription factor A in promotors with the fold change of sites of transcription factor B in promotors.

The difference becomes apparent when there is no observed overlap between the *segments of interest* and the *annotations* to be compared. In order to avoid division by 0, gat adds a pseudocount of 1 to observed and expected values: `fc = (observed + 1) / (expected + 1)`. With no observed overlap, this becomes `fc = (1 / expected + 1 )`. The amount of expected overlap correlates with the number and size of segments in the *segments of interest*. If there are more sites for A than for B, the expectation of overlap is larger for A than for B. Thus, even if there is no overlap in both cases, the fold change values reported will be different. In the case of multiple annotations of different sizes, the annotations with no overlap for both A and B will lie along a straight line.

There is no such bias if there is overlap between the *segments of interest* and the *annotations*. As both the observed and the expected overlap depend upon the segment size and number of segments of the *segments of interest*, the effect cancels itself out.

The plot below shows the log2 fold change values for the same *annotations* between two sets of *segments of interest* (CD4 and CD14).



The clouds on the upper left and lower right correspond to annotations which have no overlap with CD4 but with CD14, and vice versa. The cloud around the origin of the plot shows fold changes where both overlap for CD4 and CD14 is observed. There is no bias.

The bias can be corrected by applying a constant factor that reflects the difference in the segment sizes between the two *segments of interest*.

Note that the pseudo-count method works well when comparing fold depletion within a single set *segments of interest*. Here, the intuition is that no overlap with a larger set of annotations should give higher fold depletion than if there is no overlap with a small set of annotations.

It is possible to swap the *segments of interest* with the *annotations*. However, there is a down-side. The time consuming step in gat is the randomization of the *segments of interest*. Thus it is benefical to test few *segments of interest* against many *annotations*. When swopped, each set in the *annotations* will be randomized separately and compared to a single set of *segments of interest*.

Note that a *P-value* can be prone to misinterpretation. In particular, the *P-value* only indicates if an observed overlap is statistically significant different from the expectation. The *P-value* makes no inferences about the size of the effect and if it is biologically consequential. In particular, with increasing sample size, the expectation can be measured with higher accuracy leading to smaller differences to be detectable.

Effect size

The effect size is a measure of the strength of a phenomenom. In the context of gat a useful measure of the effect size makes use of the number of segments explained by an overlap between the *segments of interest* and a particular

annotation.

For example, an association between transcription factor binding sites and a particular annotation is likely to be more believable if it explains 20% of all transcription factor binding sites, compared to one which explains only 1% of all transcription factor binding sites. Similarly, an association that accounts for 80% of all annotations is likely to more effectual than one that accounts for only 1% of annotations.

Difference between fold changes

Gat computes assigns a statistical signficance to a fold change. The *P-value*, adjusted for multiple testing, reports the chance of observing the same or more extreme fold change given a neutral model. When using multiple *annotations*, it is thus meaningful to report the *annotations* for which statistically significant enrichment was found.

However, the *P-value* does not permit to make any inferences about the difference between two fold changes. For example, we might find that a transcription factor is 2-fold statistically enriched in promotors and 3-fold in UTRs, but we can not say with statistical confidence that the enrichment in UTRs is larger than in promotors.

Similarly, we might find that transcription factor A is 2-fold statistically enriched in promotors and transcription factor B is 3-fold enriched in promotors. Again, we can not say with statistical confidence that there is a difference in promotor binding between the two transcription factors.

The latter case, in which we have two different *segments of interest* and we want to contrast their fold enrichment against the same annotations is implemented in gat using the `gat-compare.py` command.

Briefly, the `gat-compare.py` command takes the observed and simulated counts from one or more `gat-run.py` runs. For each sample, it computes a fold change ratio as rfc = fc1 / fc2 providing a distribution of expected fold change ratios. The tool checks if the null hypothesis of no fold change difference (rfc = 1) can be rejected.

Conceptually, this works by running two simulations in-sync, one for trancription factor A and one for transcription factor B. At each iteration, the fold change is computed for each annotation between the sampled segments and the observed segments. At the same time, the fold change difference in the fold change between set A and set B is recorded. Thus, at the end of the simulations gat gets a distribution of expected fold change differences between samples and computes an empirical P-Value for the observed fold change difference between A and B.

Performance

Memory usage

GAT is limited by the amount of memory available. Factors affecting GAT's memory usage are:

- the number of segments. Memory requirements grow linearly with the number of sets and the number of intervals in the sets *segments of interest*, *annotations* and *workspace*. The total number of segments
- the number of samples. For each sample, statistics on nucleotide overlap are being kept. Thus memory requirements grow linearly with the number of samples.

If memory consumption is problematic, the following might help:

- Only working with one set of *segments of interest*.
- Only working with few sets of *annotations*.
- In a pairwise comparison, use the smaller set as *segments of interest* and the larger as *annotations*.
- Avoiding fragmentation of the workspace.
- Caching of samples

Memory consumption is usually not problematic and for typical sets reaches a few Gigabytes. If memory consumption explodes it is usually a problem with the input adat.

GAT has not been optimized for memory usage. If memory consumption is a problem, please contact the developers.

Runtime performance

As with memory usage, run-time performance of GAT is linearly related to the number of segments and the number of samples.

- the number of segments in the *segments of interest*. More segments require more sampling steps. Usually, an input size with twice as many segments will take twice as long. Runtime behaviour might be worse in extreme cases where the sampler has difficulties placing the last segments, for example if the segment density is high.
- the number of samples. Each additional sample will require an additional amount of time.
- the number of segments in the *annotations*. Usually less of a factor, but it becomes a factor if overlap statisticts need to be computed for many different annotations. In this case, generating a single sample might be quicker than computing overlap statisticts of that sample with

As runtime performance is a linear function of most variables, runtime can be reduced by using multiple CPUs or cores (see *Using multiple CPU/cores*).

Sampling performance

In order to check for biases in the sampling procedure, we run automated tests to check for even coverage of nucleotides by the sampler and absence of edge effects.

Background

History

This module has been inspired by the TheAnnotator tool first used in the analysis by Ponjavic et al (2007) by Gerton Lunter and early work of Caleb Webber.

The differences are:

- permit more measures of association. The original Annotator used nucleotide overlap, but other measures might be useful like number of elements overlapping by at least x nucleotides, proximity to closest element, etc.
- easier user interface and using standard formats.
- permit incremental runs. Annotations can be added without recomputing the samples.
- faster.

Comparison to other methods

Testing for the association between genomic features is a field of long-standing interest in genomics and has gained considerable traction with the publication of large scale genomic data sets such as the ENCODE data.

Generally we believe that the problem of testing for association has not been fully resolved and advise every genomicist to apply several methods. The list of tools/services below is not exhaustive:

GREAT (MacLean et al. (2010)) uses a binomial test to test if transcription factor binding sites are associated with regulatory domains of genes. GREAT has a convenient web interface with many annotations pre-loaded. Compared to GREAT, GAT can measure depletion and can

GenometriCorr (Favorov et al. (2012)) compute a variety of distance metrics when comparing two interval sets and then apply a set of statistical tests to measure association. GenomicCorr is a good exploratory tool to generate hypotheses about the relationships of two genomic sets of intervals. Compared to GenometriCorr, GAT can simulate more realistic genomic scenarios, for example, segments might not occur in certain regions (due to mapping problems) or occur at reduced frequency (G+C biases).

The GSC (The Encode Project Consortium (2012)) metric (for Genome Structure Correlation) is inspired by the analysis of approximately piecewise stationary time series . The GSC metric estimates the significance of an association metric by estimating the random expectation of the association metric using randomly chosen intervals on the genome. This expectation is then used to test if the observed value of the metric (nucleotide overlap, region overlap, ...) is higher than expected. The method is described in the supplemental details of the first and recent ENCODE papers (*Birney et al. (2007)_*, ...) and here.

BITS (Layer et al. (2013)) (Binary Interval Search) is a method to perform quick overlap queries between genomic data sets. It implements a Monte-Carlo method for simulation that is particularly suited towards making all on all comparisons between a large number data sets.

Benchmark

We used the example from *Tutorial - Interval overlap* to perform a rough comparison between various methods. In all cases, we used `n = 1000` for simulations. Times are wall-clock times. Please note that this is not a rigorous benchmark.

Method	Set1	Set2	Observed	Expected	P-value	Time
BITS	srf	jurkat	450	5.24	<0.001	43s
BITS	srf	hepg2	381	9.87	<0.001	39s
BITS	srf	hepg2/jurkat	9	5.7	0.13	28s
BITS	jurkat	hepg2	47237	3548	<0.001	106s
GSC	srf	jurkat			0.0004	58s
GSC	srf	hepg2			6.9E-11	54s
GSC	srf	hepg2/jurkat			5.23E-7	40s
GSC	jurkat	hepg2			0	159s
GAT	srf	jurkat	20183	247.6	<0.001	11s
GAT	srf	hepg2	18965	601.4	<0.001	11s
GAT	srf	hepg2/jurkat	425	327.3	0.21	11s
GAT	jurkat	hepg2	6163503	457332.8	<0.001	316s

BITS and GAT are fairly comparable, even though they use different metrics for the association (number of segments overlapping versus number of nucleotides overlapping). GAT is quicker on smaller data sets, while BITS outperforms on large datasets.

GSC reports a significant association in the comparison between srf and dhs intervals specific to hepg2 cells, while the other two tools do not, which is the biologically plausible result. It is difficult to say if there indeed is an association, or GSC is overestimating association.

Glossary

Interval a (genomic) segment with a chromosomal location (contig,start and end).

Intervals a set of one or more intervals.

workspace the genomic regions accessible for simulation.

annotations sets of intervals annotating various regions of the genome.

segments of interest sets of intervals whose association is tested with *annotations*

sampled segments randomized versions of the *segments of interest*. A set of sampled segments is one *sample*.

sample one set of *sampled segments*.

isochore Usually, isochores are defined as genomic regions of homogeneous G+C content. In GAT, isochores can mean any genomic regions with a shared property. Isochores are used to eliminate the effect of confounders in an analysis.

isochores plural of *isochore*.

isochore workspaces the workspace split according to *isochores*.

bed an interval format. Intervals are in a tab-separated list denoted as `contig`, `start`, `end`. A bed file can contain several *tracks* which will be treated independently. Tracks are either delineated by a line of the format: `track name=<trackname>` or by an optional fourth column (field `name`). See UCSC for more information about bed files.

fold fold change, computed as the ratio of observed over expected

p-value significance of association. The P-value is an estimate of the probability to obtain an observed (or larger) overlap between two segment sets by chance.

q-value multiple testing corrected p-value. The qvalue can either be computed using the q-value procedure suggested by Storey et al. (2002) or using other FDR approaches such as Bonferroni, Benjamini-Hochberg, etc. implemented in the R function `p.adjust`.

pvalue the column containing the *p-value*.

qvalue the column containing the *q-value*.

track a set of segments within a *bed* formatted file. Tracks are either grouped using the `track name=<trackname>` prefix or by an optional fourth column (the `name` column).

annotation a genomic annotation, which is a collection of intervals.

observed the observed value of a metric.

expected the expected value of a metric as determined by simulations.

fold the fold change, defined as the ratio of observed over expected.

l2fold the logarithm (base 2) of *fold* change.

tracks plural of *track*

samples a set of samples (see *sample*)

Release Notes

1.3.5:

- Bugfixes - do not delete during container iteration

1.3.4:

- Bugifxes to (hopefully) complete python 2/3 compatibility.

1.3.0:

- GAT now works under python 2 and python 3
- #3: gat-compare iterates now over shared tracks, not just the 'merged' track.
- #2: do not split isochore in chrom.iso if isochore is "."

CHAPTER 3

Developers' notes

The following section contains notes for developers.

Notes

Sampling strategies

Sampling creates a new set of *interval* `P`. There are several different strategies possible.

Annotator strategy

In the original Annotator strategy, samples are created in a two step procedure. First, the length of a sample segment is chosen randomly from the empirical segment length distribution. Then, a random coordinate is chosen. If the sampled segment does not overlap with the workspace it is rejected.

Before adding the segment to the sampled set, it is truncated to the workspace.

If it overlaps with a previously sampled segment, the segments are merged. Thus, bases shared between two segments are not counted twice.

Sampling continues, until exactly the same number of bases overlap between the `P` and the `W` as do between `S` and `W`.

Note that the length distribution of the intervals in `P` might be different from `S`.

The length is sampled from the empirical histogram of segment sizes. The granularity of the histogram can be controlled by the options `-bucket-size` and `--nbuckets`. The largest segment should be smaller than `bucket-size * nbuckets`. Increase either if you have large segments in your data set, but smaller values of `nbuckets` are quicker.

This method is quick if the workspace is large. If it is small, a large number of samples will be rejected and the procedure slows down.

This sampling method is compatible with both distance and overlap based measures of associaton.

Workspaces and isochores

Workspaces limit the genomic space available for segments and annotations. Isochores split a workspace into smaller parts that permit to control for confounding variables like G+C content.

The simplest workspace is the full genome. For some analyses it might be better to limit to analysis to autosomes.

Examples for the use of isochores could be to analyze chromosomes or chromosomal arms separately.

If isochores are used, the length distribution and nucleotide overlaps are counted per isochore ensuring that the same number of nucleotides overlap each isochore in `P` and `S` and the length distributions per isochore are comparable.

Empirical length distribution

The empirical length distribution is created from all *intervals* in `S`. The full segment length is chosen even if there is partial overlap. Optionally, the segment can be truncated. From Gerton:

```
What is the best choice depends on the data. Not truncating can lead
to a biased length distribution if it is expected that segments that
only partially overlap the workspace have very different lengths. However,
truncating can lead to spurious short segments.
```

Benchmarking

This section contains quality control plots from the unit testing.

Position sampling

The following section looks at the position sampling algorithms.

Position sampling

The `TestSamplerPosition` tests if the position sampling works as expected. In particular, look out for edge effects.

Segment sampling algorithms

The following plots benchmark the segment sampling behaviour of the various segment sampling algorithms implemented in GAT.

Statistics

For 1-sized fragments (i.e. SNPs), the statistics can be checked against a hypergeometric distribution (sampling without replacement). All the tests below use a single continuous workspace of 1000 nucleotides seeded with a varying number of SNPs.

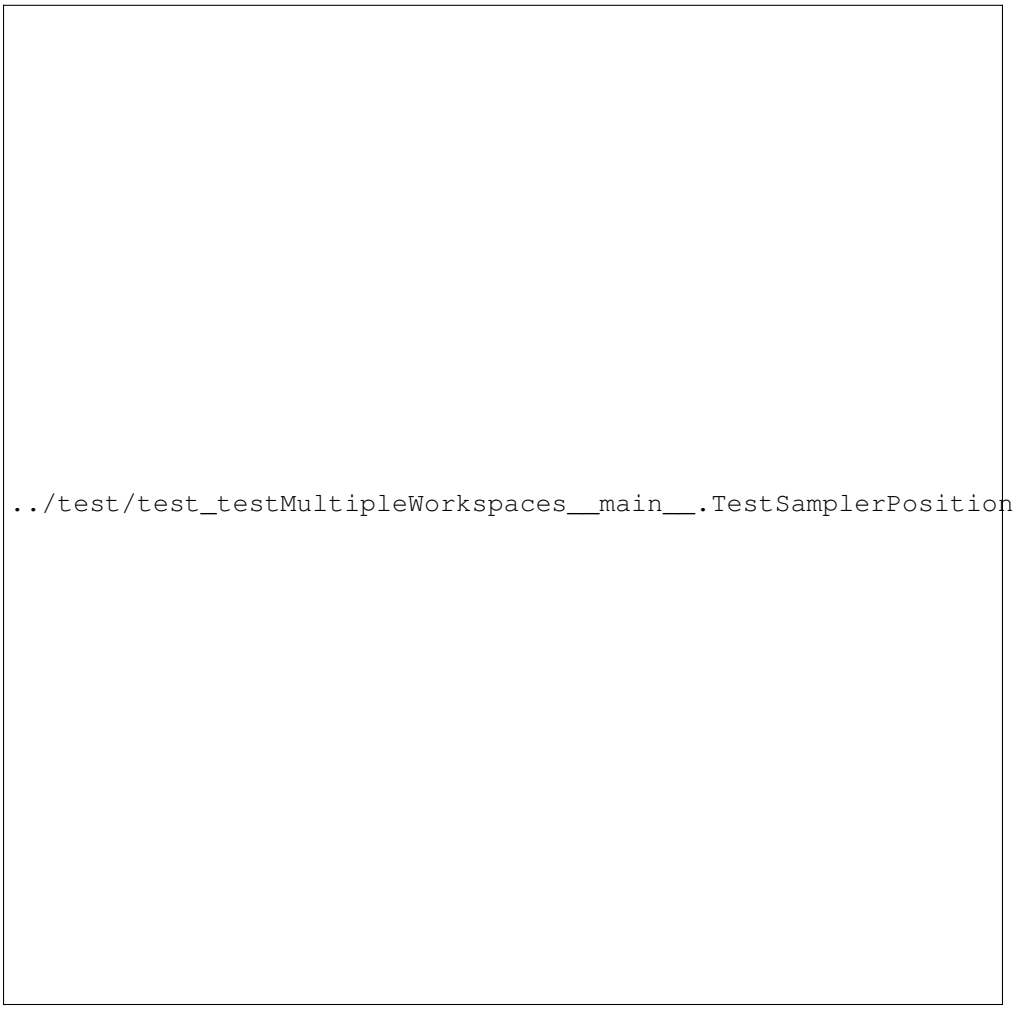


Fig. 3.1: Multiple work spaces. 10 workspaces of size 100, spaced every 1000 nucleotides

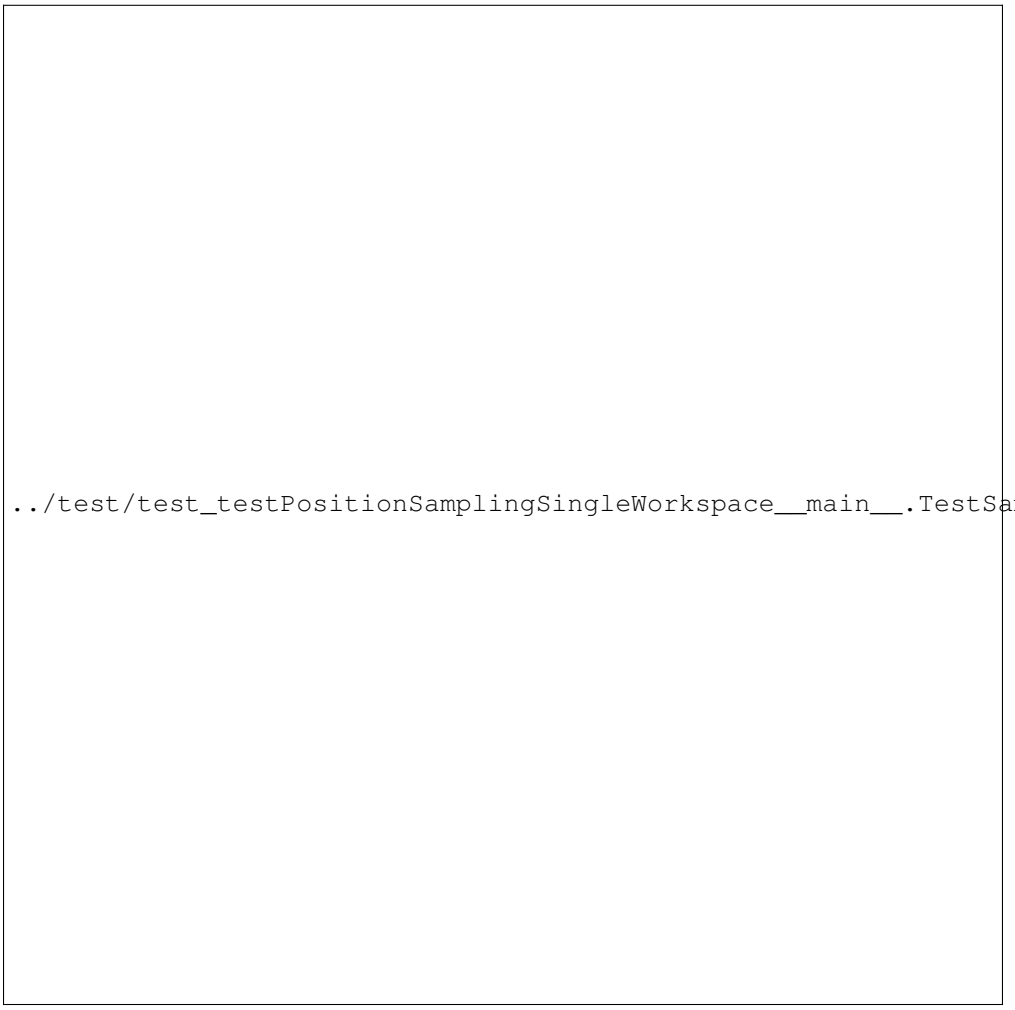


Fig. 3.2: A single work space.

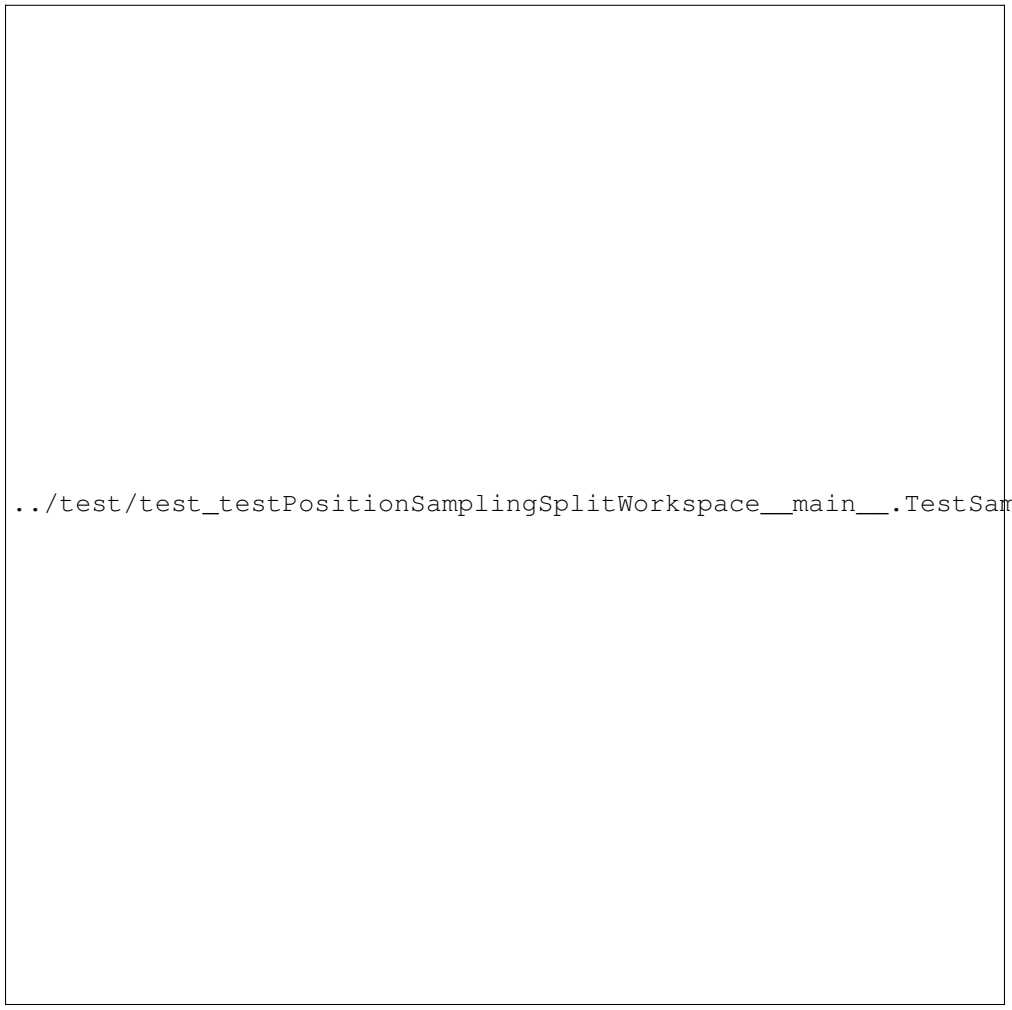


Fig. 3.3: A workspace split in the middle without a gap.

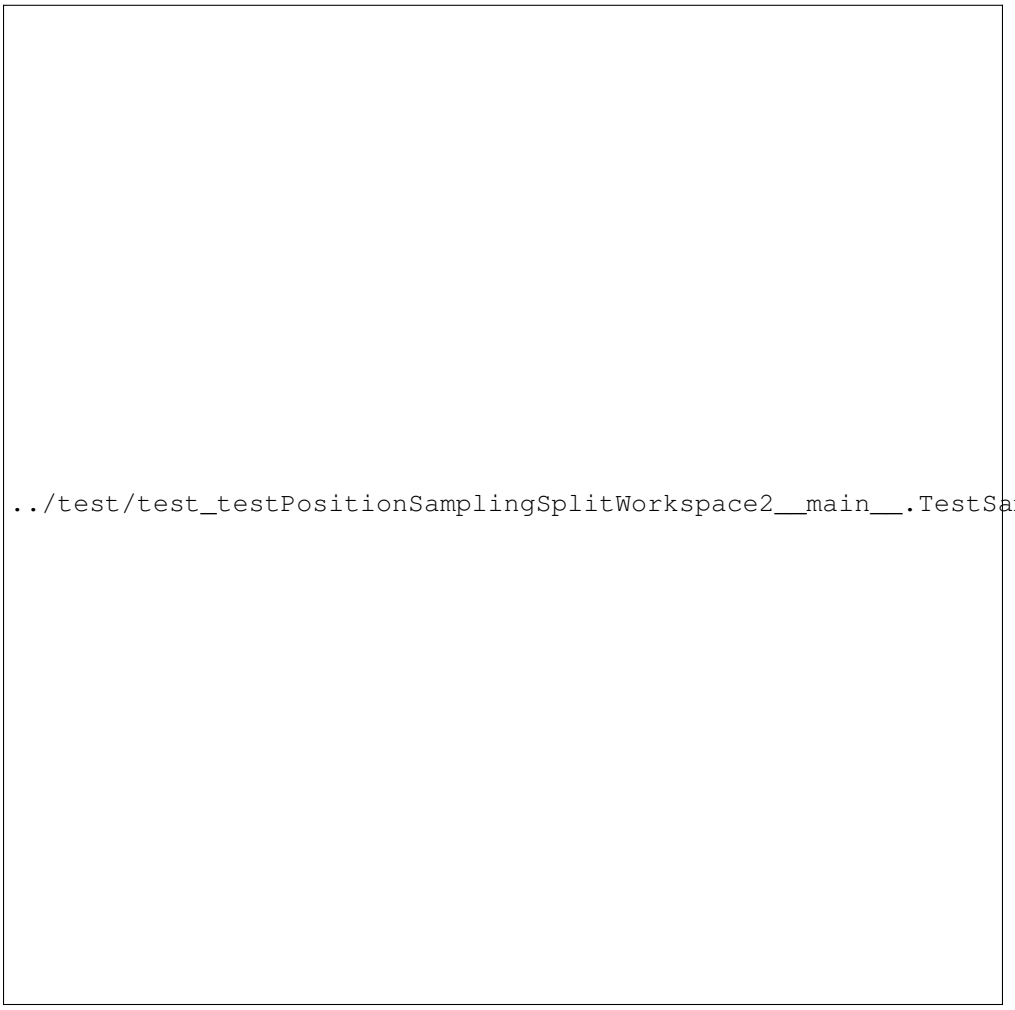


Fig. 3.4: A workspace split in the middle with a gap in between.

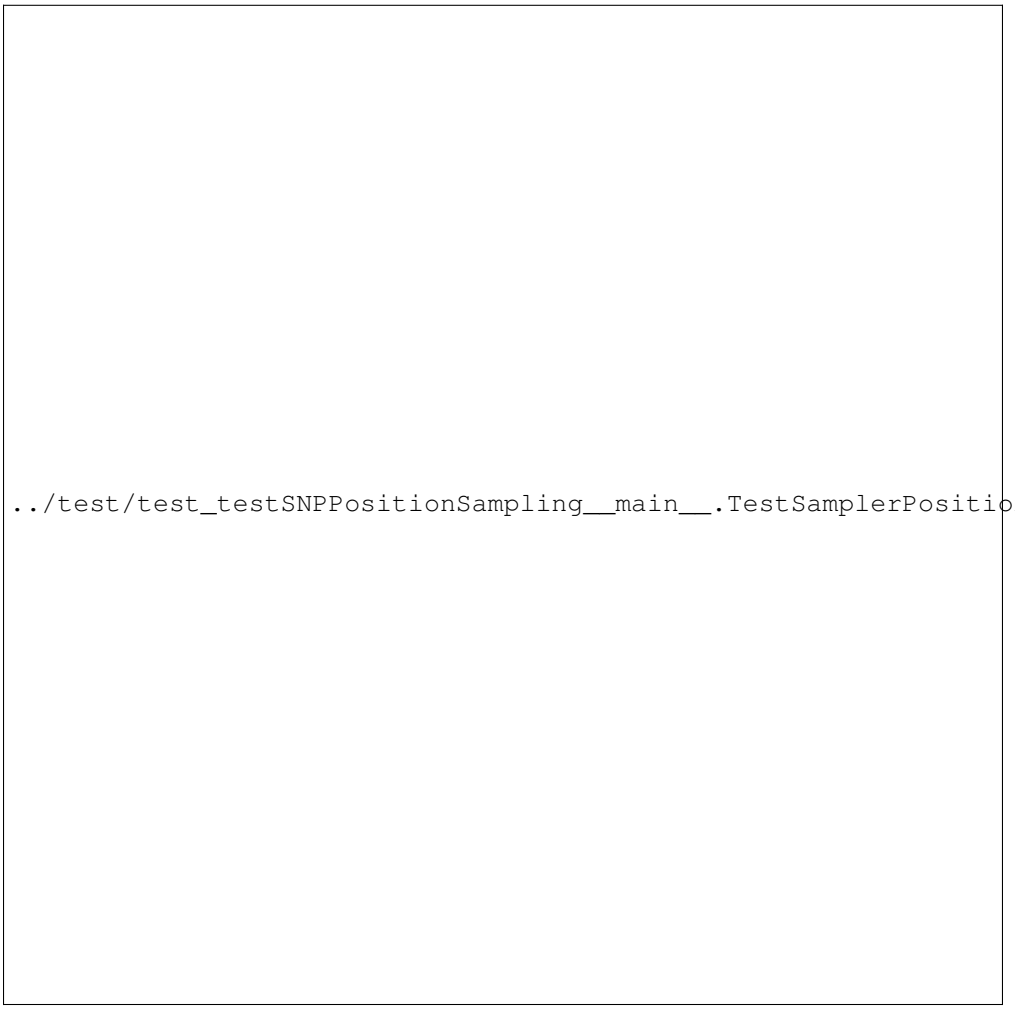


Fig. 3.5: 10 workspaces of size 100, segment size of 1 (SNP).

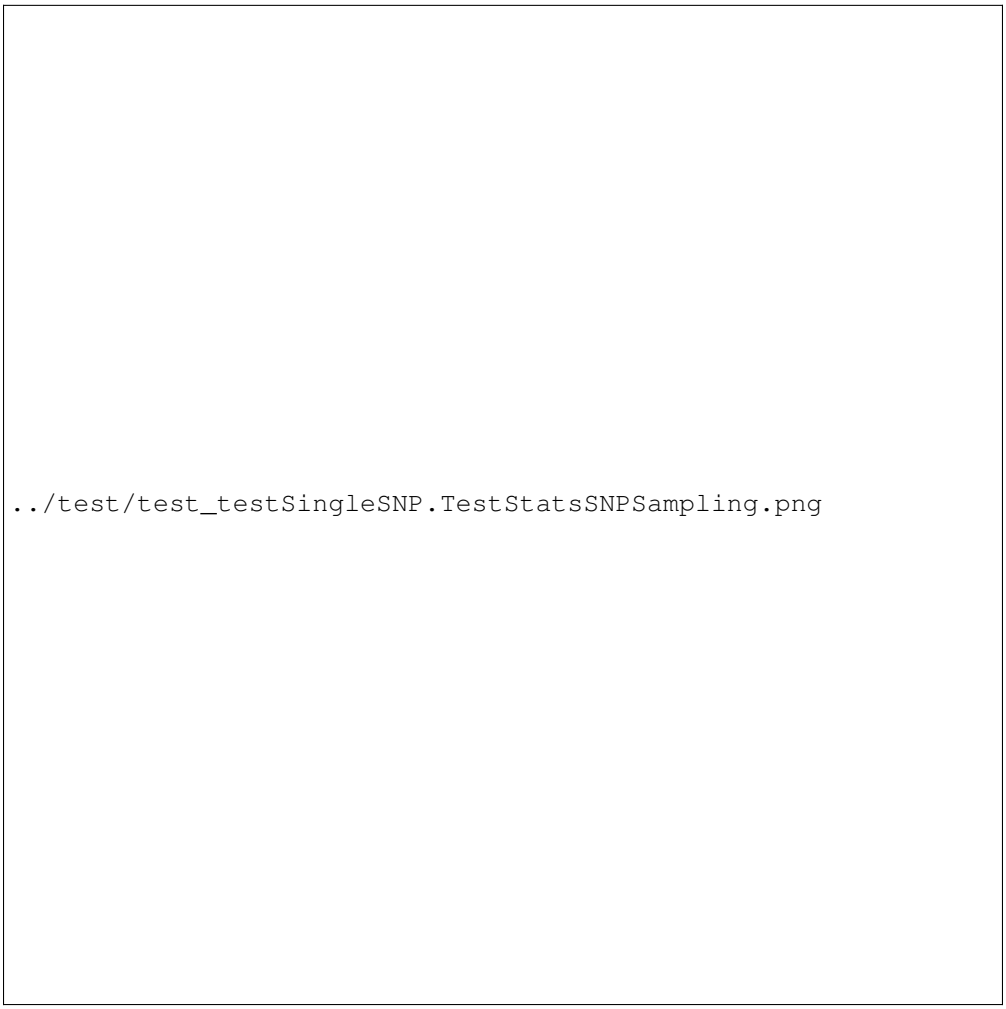


Fig. 3.6: Test with a single SNP. Here, there are no issues with multiple hits. The workspace contains a single annotation of increasing size (1,3,5,...,99)

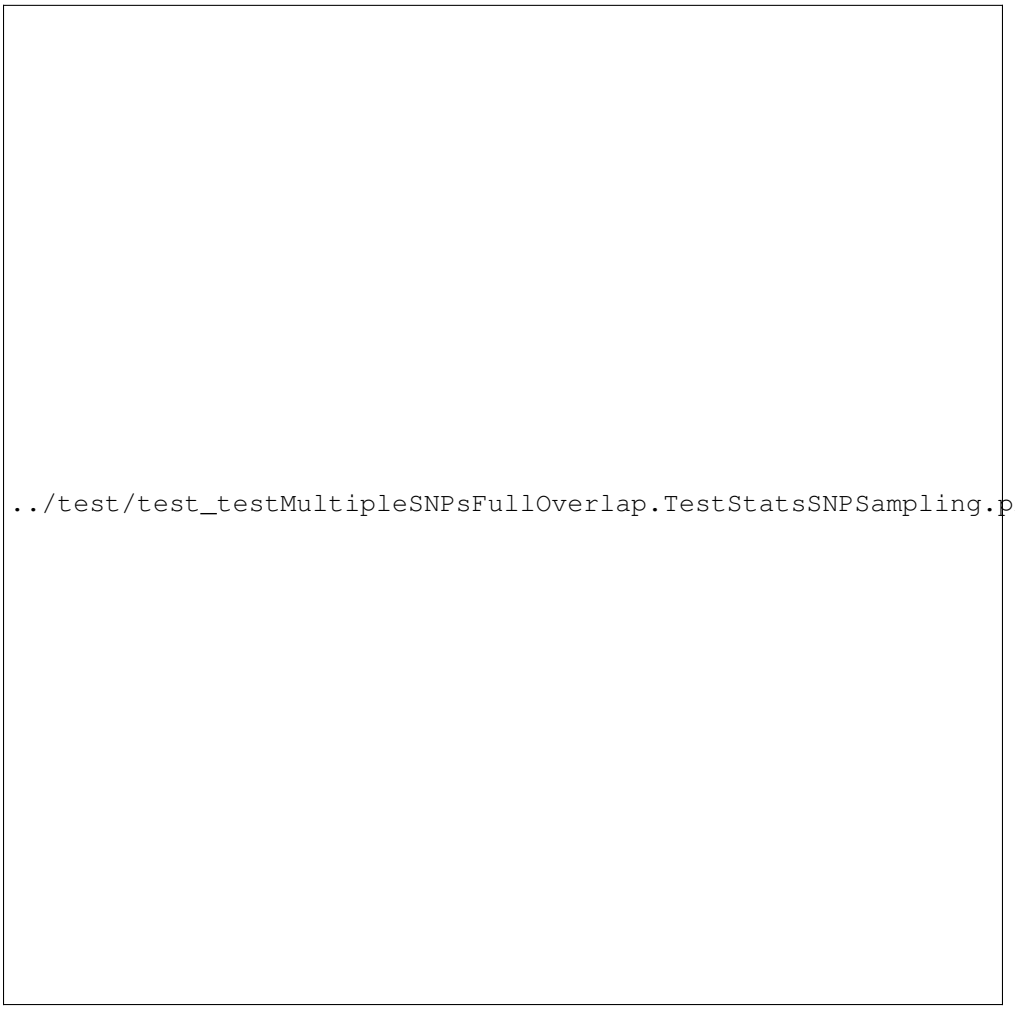


Fig. 3.7: In this test, *10* SNPs are in the segment list. The workspace contains a single annotation of size (10, 15, ..., 105). All SNPs overlap the annotated part of the workspace and hence all results are highly signficant.

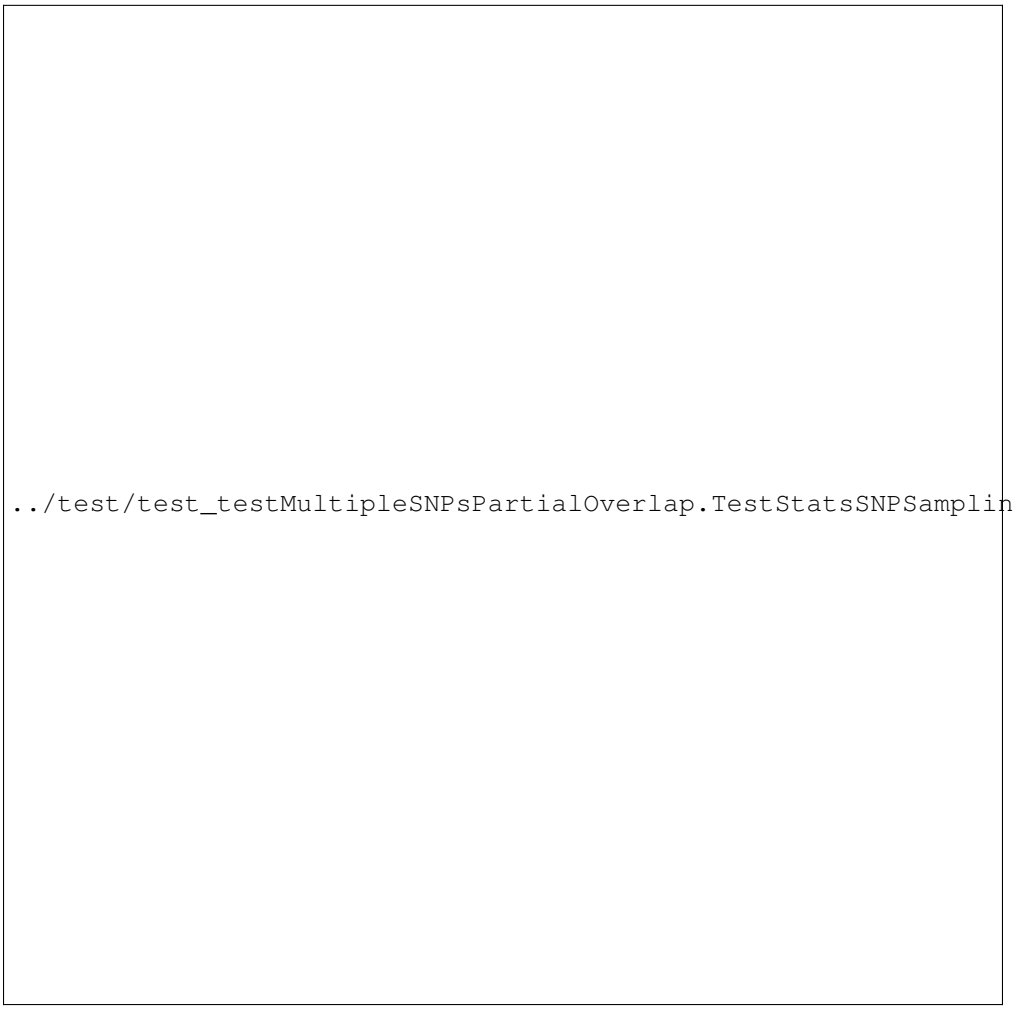


Fig. 3.8: In this test, *10* SNPs are in the segment list. The workspace contains a single annotation. Annotations are all of size *10*, but the overlap of SNPs with annotations varies from 0 to 10.

Statistics

Gat

SNPs

For 1-sized fragments (i.e. SNPs), the statistics can be checked against a hypergeometric distribution (sampling without replacement). All the tests below use a single continuous workspace of 1000 nucleotides seeded with a varying number of SNPs.

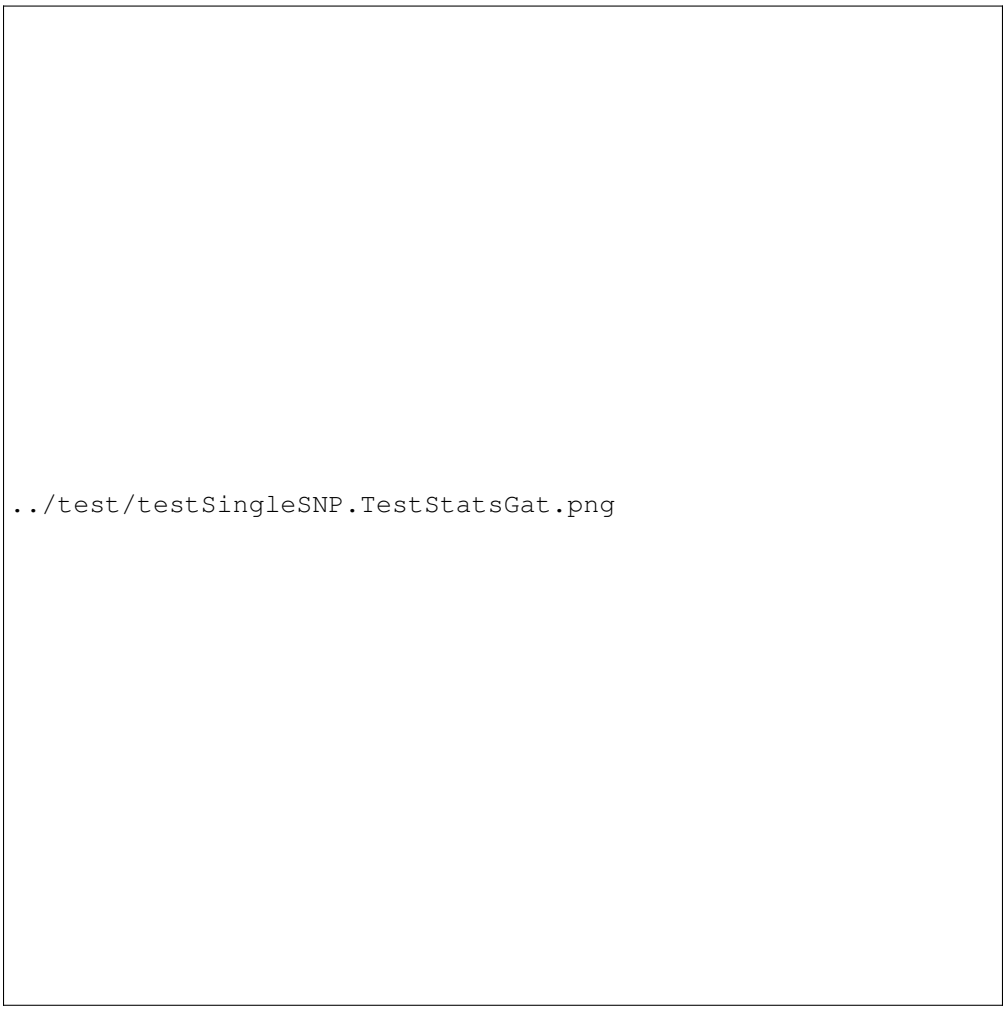


Fig. 3.9: Test with a single SNP. Here, there are no issues with multiple hits. The workspace contains a single annotation of increasing size (1,3,5,...,99)

Intervals

Annotator

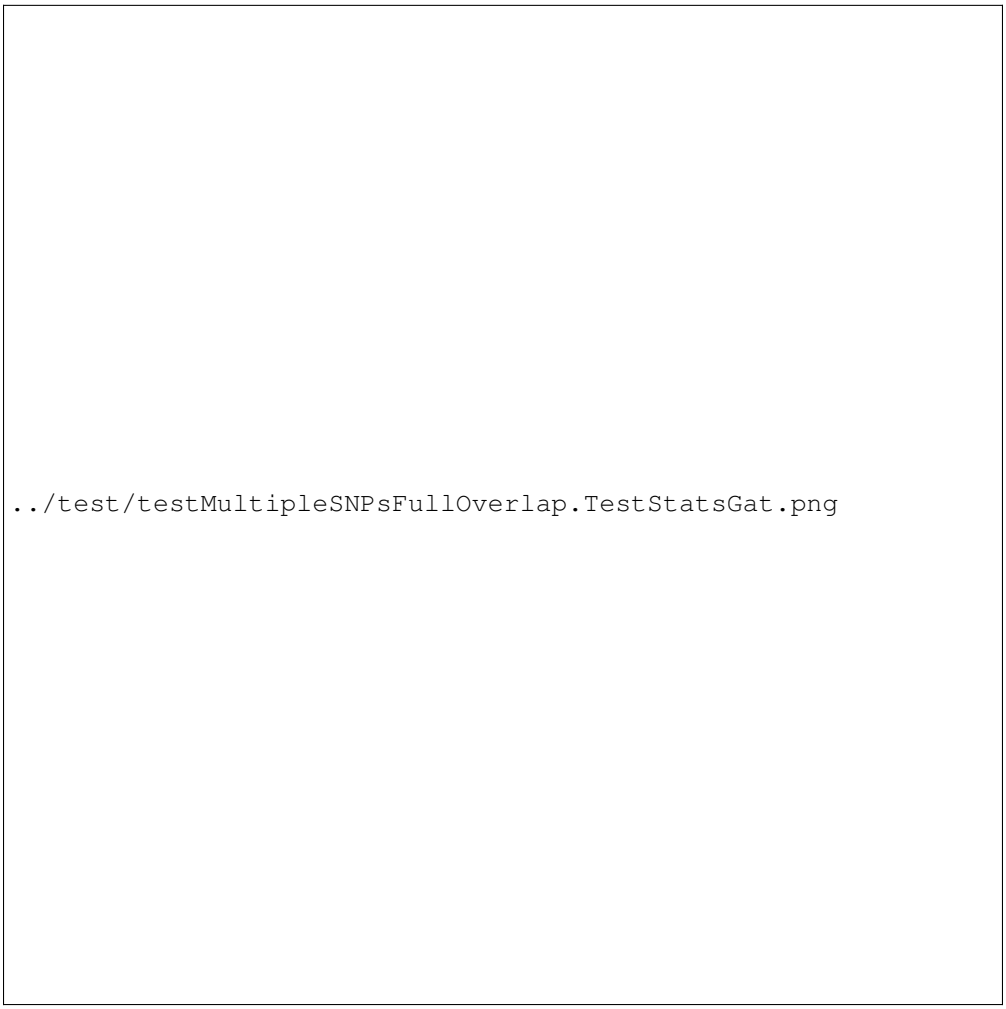


Fig. 3.10: In this test, *10* SNPs are in the segment list. The workspace contains a single annotation of size (10, 15, ..., 105). All SNPs overlap the annotated part of the workspace and hence all results are highly signficant.

Fig. 3.11: In this test, *10* SNPs are in the segment list. The workspace contains a single annotation. Annotations are all of size *10*, but the overlap of SNPs with annotations varies from 0 to 10.

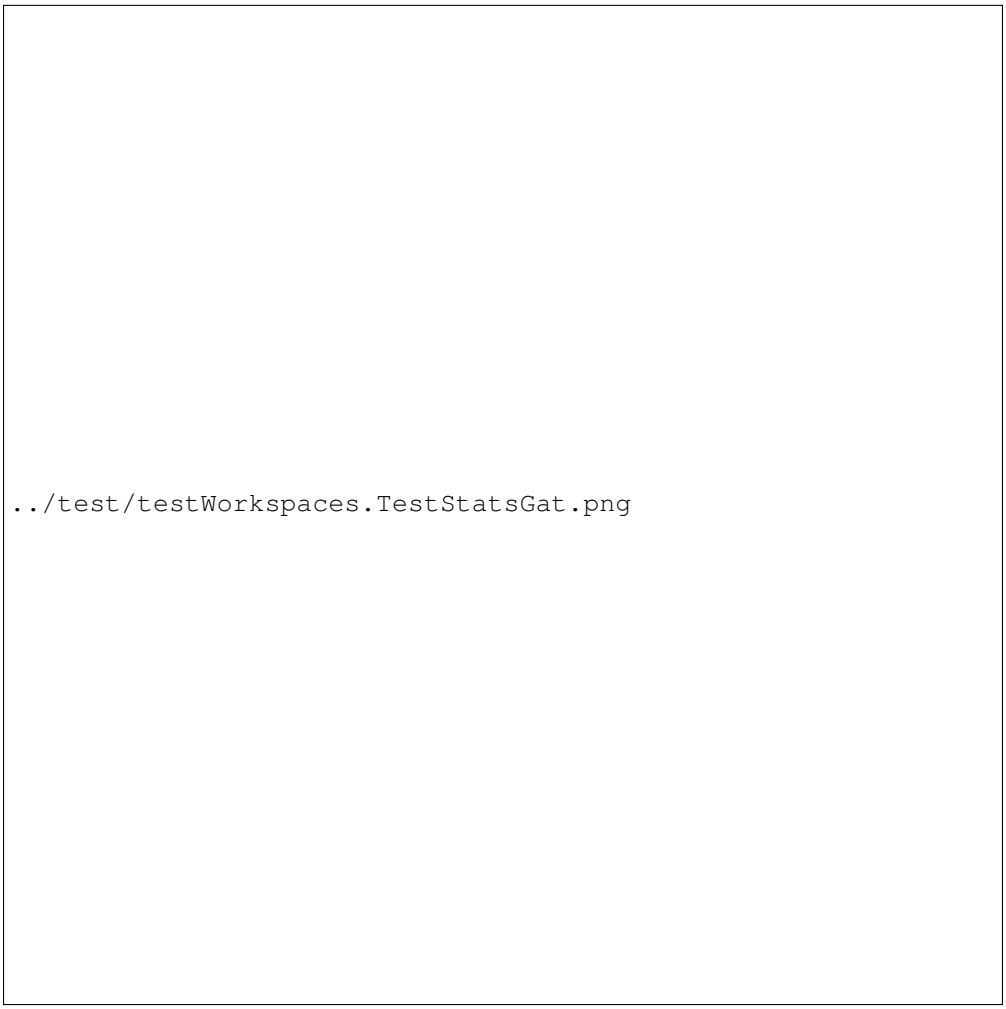


Fig. 3.12: workspace = 500 segments of size 1000, separated by a gap of 1000 annotations = 500 segments of size 1000, separated by a gap of 1000, shifted up 100 bases segments = a SNP every 100 bp

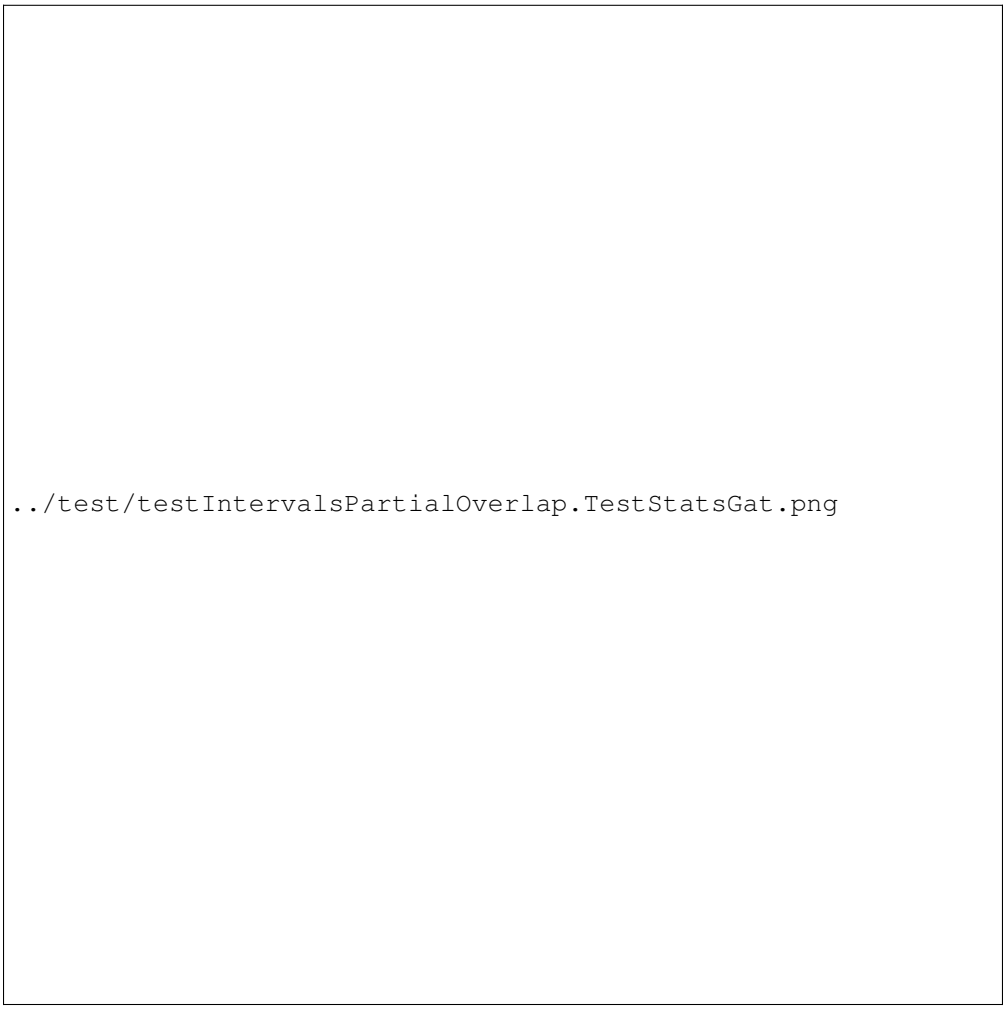


Fig. 3.13: In this test, there is one segment of size *100*. Annotations are of size *100* with decreasing overlap.

SNPs

For 1-sized fragments (i.e. SNPs), the statistics can be checked against a hypergeometric distribution (sampling without replacement). All the tests below use a single continuous workspace of 1000 nucleotides seeded with a varying number of SNPs.

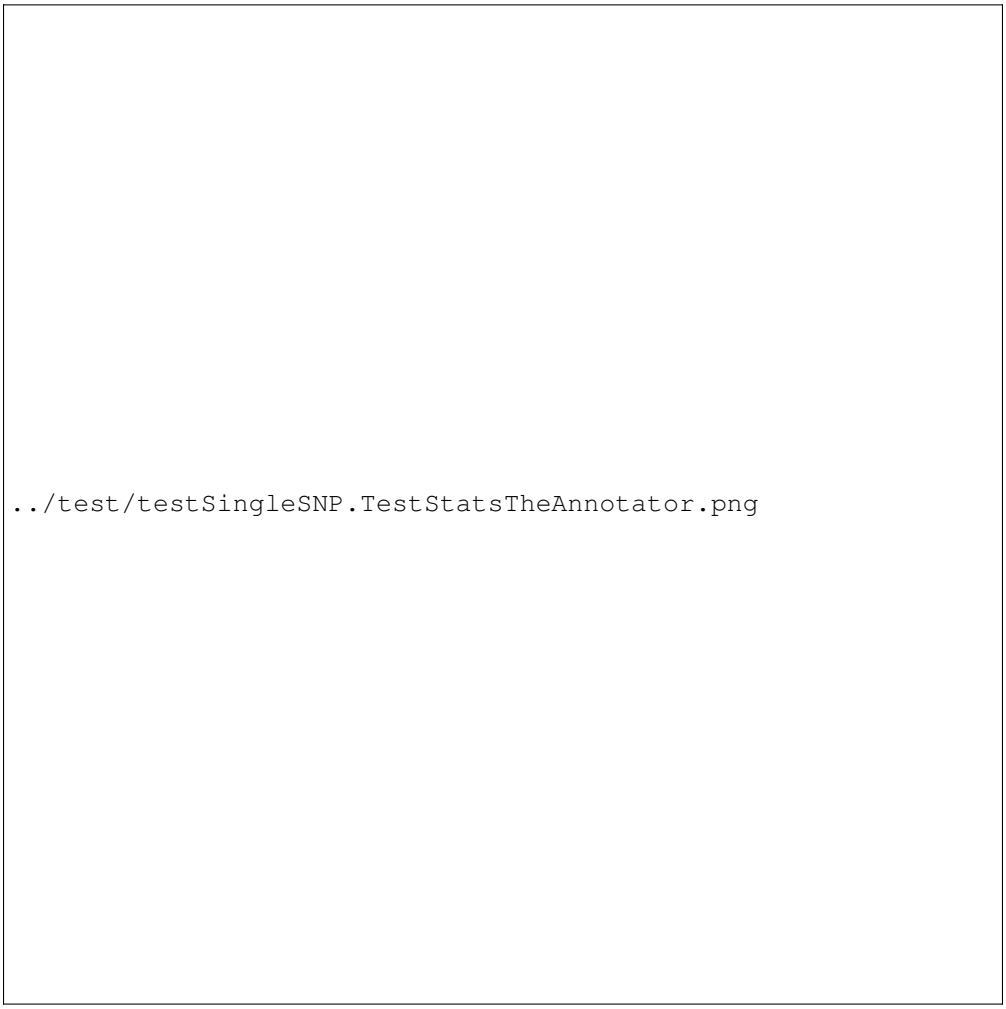


Fig. 3.14: Test with a single SNP. Here, there are no issues with multiple hits. The workspace contains a single annotation of increasing size (1,3,5,...,99)

Intervals

Simulation algorithms

Gat implements a variety of simulation algorithms. Not all of them are of practical use.

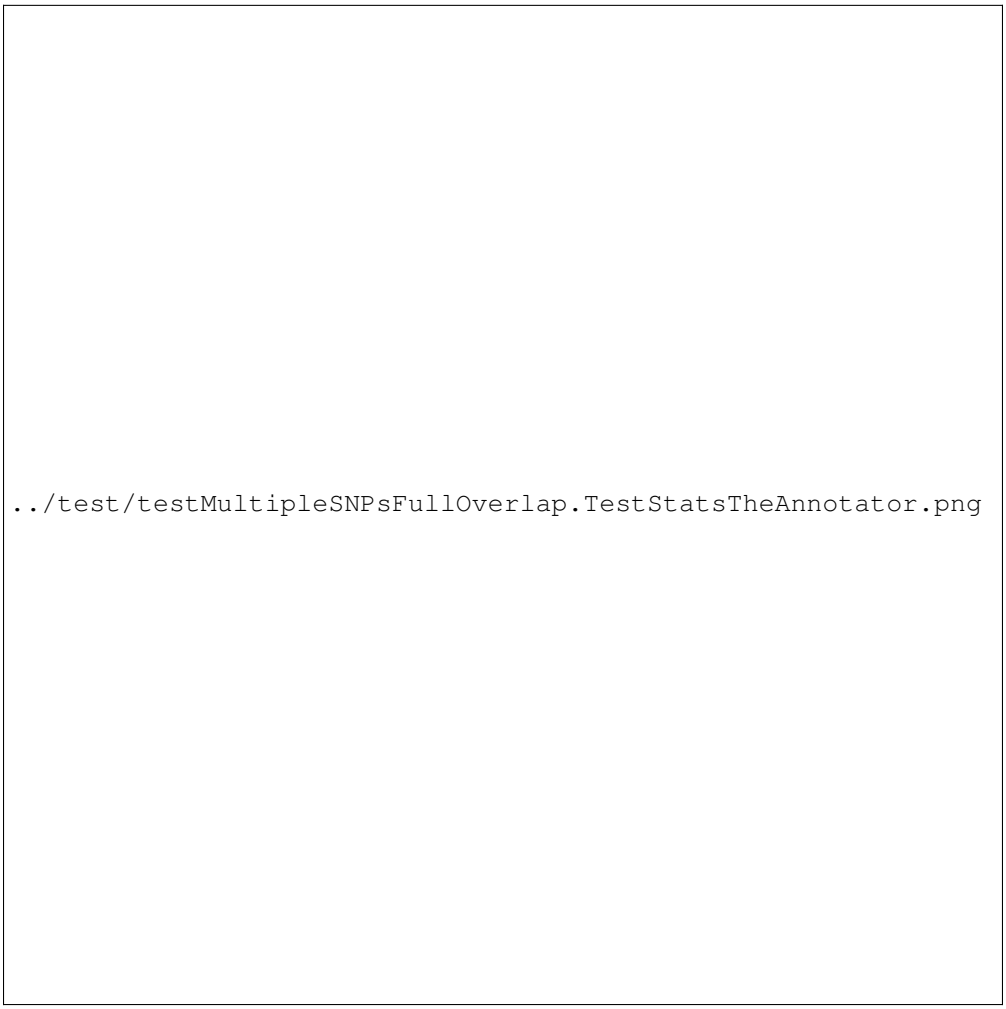


Fig. 3.15: In this test, *10* SNPs are in the segment list. The workspace contains a single annotation of size (10, 15, ..., 105). All SNPs overlap the annotated part of the workspace and hence all results are highly signficant.

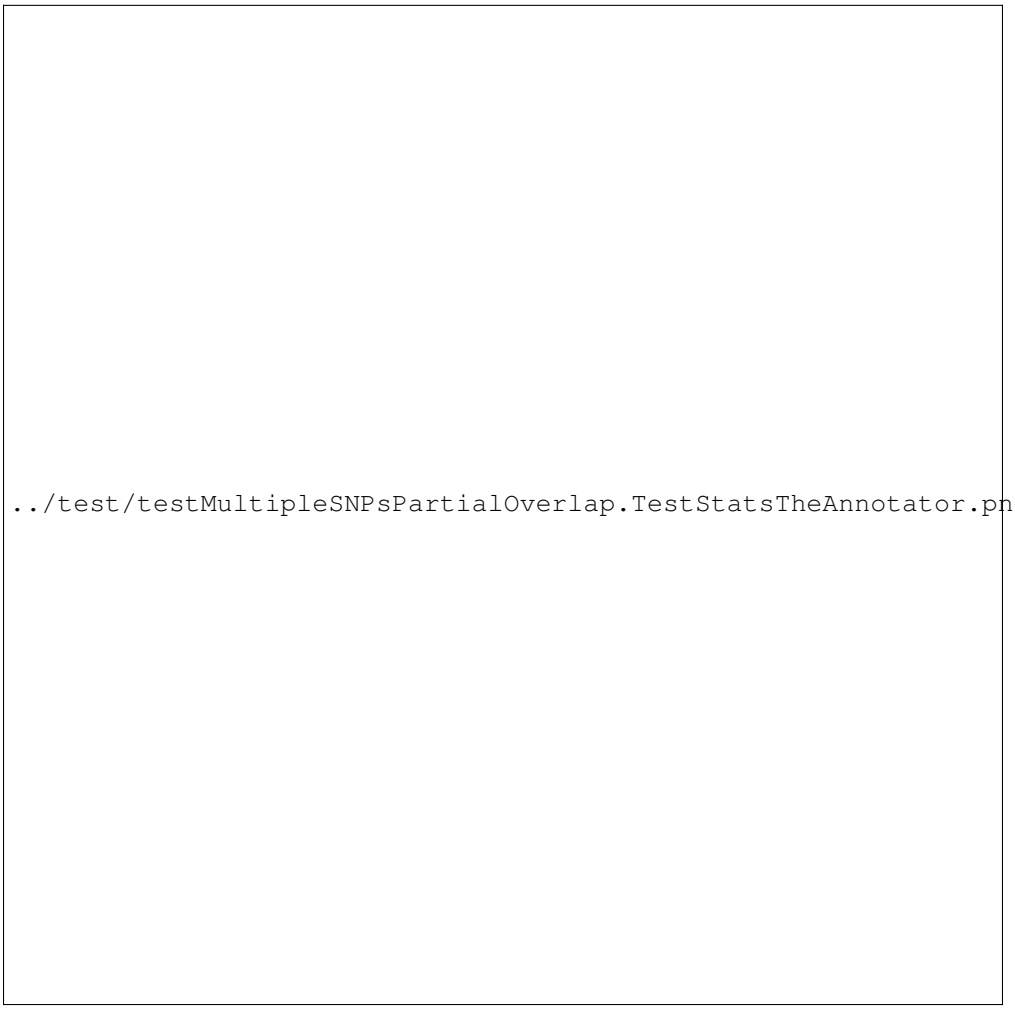


Fig. 3.16: In this test, *10* SNPs are in the segment list. The workspace contains a single annotation. Annotations are all of size *10*, but the overlap of SNPs with annotations varies from 0 to 10.

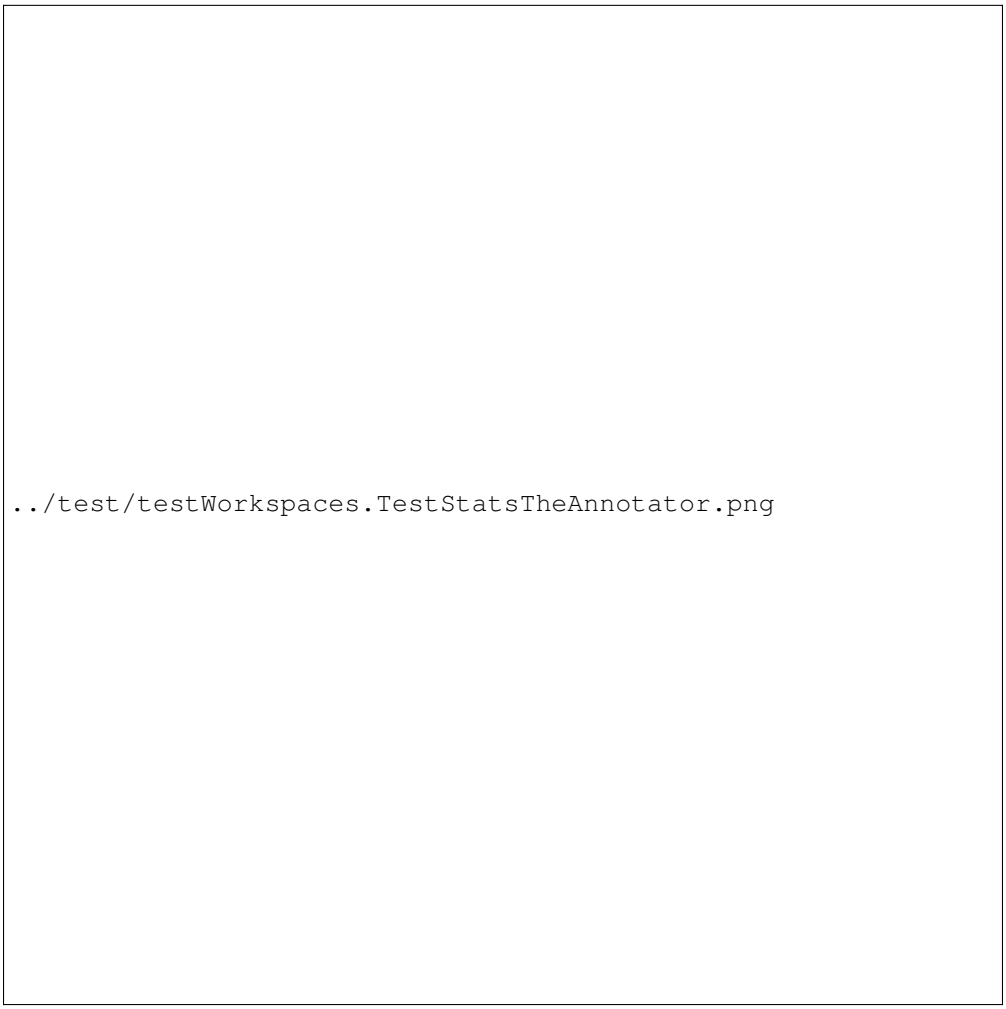


Fig. 3.17: workspace = 500 segments of size 1000, separated by a gap of 1000 annotations = 500 segments of size 1000, separated by a gap of 1000, shifted up 100 bases segments = a SNP every 100 bp

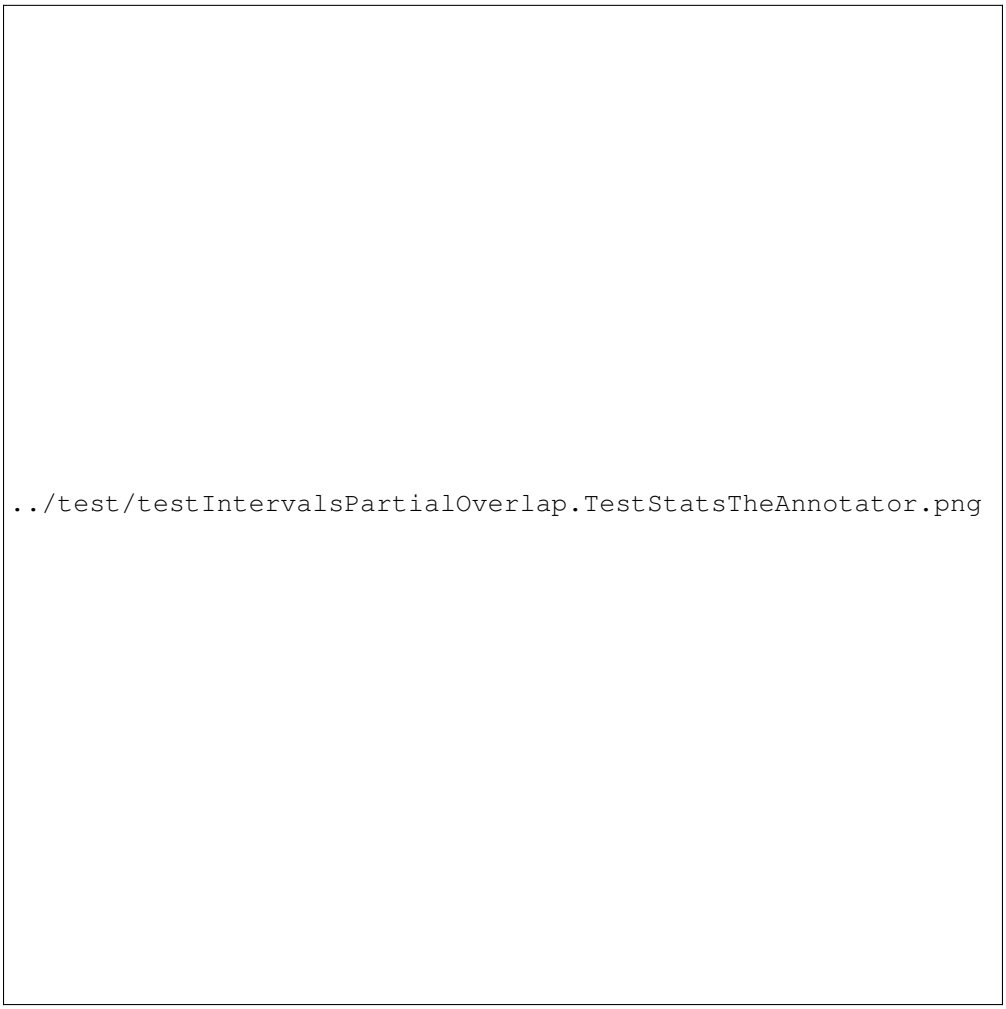


Fig. 3.18: In this test, *10* SNPs are in the segment list. The workspace contains a single annotation. Annotations are all of size *10*, but the overlap of SNPs with annotations varies from 0 to 10.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index