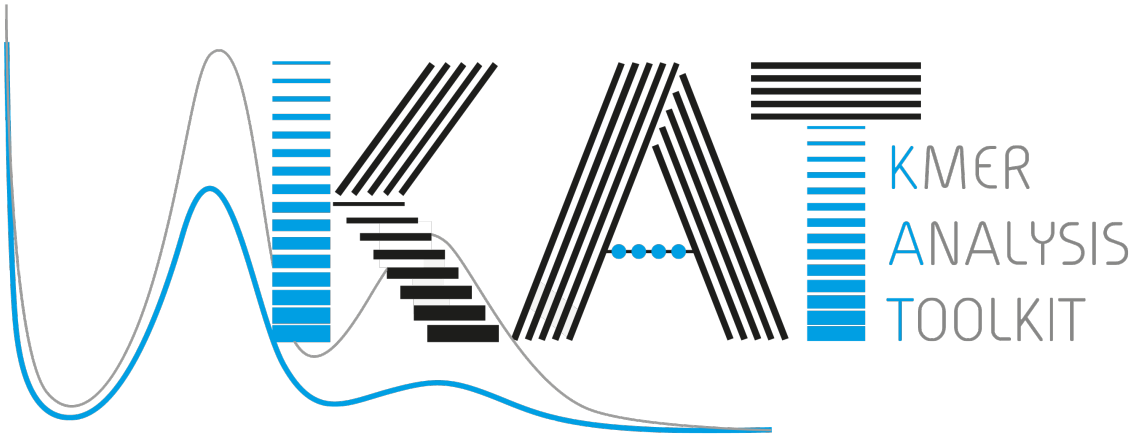


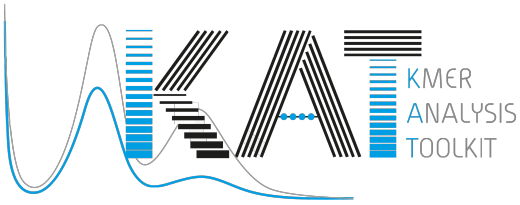
KAT Documentation

Release 2.4.2

Daniel Mapleson
George Kettleborough
Bernardo Clavijo
Gonzalo Garcia
Jon Wright

Aug 15, 2018

Contents



KAT provides a suite of tools that, through the use of k-mer counts, help the user address or identify issues such as determining sequencing completeness for assembly, assessing sequencing bias, identifying contaminants, validating genomic assemblies and filtering content. KAT is geared primarily to work with high-coverage genomic reads from Illumina devices, although can work with any fasta or fastq sequence file.

At it's core KAT exploits the concept of k-mer spectra (histograms plotting number of distinct k-mers at each frequency). By studying properties of the k-mer spectra it's possible to discover important information about the data quality (level of errors, sequencing biases, completeness of sequencing coverage and potential contamination) and genomic complexity (size, karyotype, levels of heterozygosity and repeat content). Further information can be gleaned through pairwise comparison of spectra, making KAT useful for WGS library comparisons and assembly validation.

The K-mer counting itself, a critical element for all KAT tools, is accomplished through an integrated and modified version of Jellyfish2's counting method. We selected Jellyfish for this task because it supports large K values and is one of the fastest k-mer counting programs currently available.

CHAPTER 1

Installation

1.1 From brew

If you have brew installed on your system you should be able to install a recent version of KAT by simply typing:

```
brew install brewsci/bio/kat
```

Many thanks to @sjackman for this one!

1.2 From bioconda

If you use bioconda you can install KAT using :

```
bioconda install kat
```

1.3 From source

If you wish to install KAT from source, because you don't have brew installed, or wish to ensure you have the latest version, first ensure these dependencies are installed and configured on your system:

- **GCC** V4.8+
- **make**
- **autoconf** V2.53+
- **automake** V1.11+
- **libtool** V2.4.2+
- **pthreads** (probably already installed)
- **zlib**
- **Python** V3.5+ with the *tabulate*, *scipy*, *numpy* and *matplotlib* packages and C API installed. This is optional but highly recommended, without python KAT functionality is limited: no plots, no distribution analysis, and no documentation.
- **Sphinx-doc** V1.3+ (Optional: only required for building the documentation.

NOTE ON INSTALLING PYTHON: Many system python installations do not come with the C API immediately available, which prevents KAT from embedding python code. We typically would recommend installing anaconda3 as this would include the latest version of python, all required python packages as well as the C API. If you are running a debian system and the C libraries are not available by default and you wish to use the system python installation the you can install them using: `sudo apt-get install python-dev`. Also if you are using a python installation outside your system directory, please make sure you have your PATH and LD_LIBRARY_PATH (or LD_RUN_PATH) environment variables set appropriately.

Then proceed with the following steps:

- Clone the git repository (For ssh: `git clone git@github.com:TGAC/KAT.git`; or for https: `git clone https://github.com/TGAC/KAT.git`), into a directory on your machine.
- Change directory into KAT project: `cd KAT`
- Build boost (this may take some time): `./build_boost.sh`
- Setup the KAT configuration scripts by typing: `./autogen.sh`.
- Generate makefiles and confirm dependencies: `./configure`. The configure script can take several options as arguments. One commonly modified option is `--prefix`, which will install KAT to a custom directory. By default this is `/usr/local`, so the KAT executable would be found at `/usr/local/bin` by default. Python functionality can be disabled using `--disable-pykat`. Type `./configure --help` for full list of options. Please check the output to ensure the configuration is setup as you'd expect.
- Compile software: `make`. You can leverage extra cores duing the compilation process using the `-j <#cores>` option. Also you can see all command lines used to build the software by setting `V=1`.
- Run tests (optional) `make check`. (The `-j` and `V=1` options described above are also supported here.)
- Install: `make install`. If you've not provided a specific installation directory, you will likely need to prefix this command with `sudo` in order to provide the permissions required to install to `/usr/local`.

If sphinx is installed and detected on your system then html documentation and man pages are automatically built during the build process. If it is not detected then this step is skipped. Should you wish to create a PDF version of the manual you can do so by entering the `doc` directory and typing `make pdf`, this is not executed by default.

NOTE: if KAT is failing at the `./autogen.sh` step you will likely need to install autotools. The following command should do this on MacOS: `brew install autoconf automake libtool`. On a debian system this can be done with: `sudo apt-get install autoconf automake libtool`.

Python scripts

KAT will install some python scripts to your `<prefix>/bin` directory. If you selected a custom location for prefix and wish to access these scripts directly, then it may be necessary to modify your $PYTHONPATH environment variable. Ensure that `<prefix>/lib/python<version>/site-packages`, is on your PYTHONPATH, where <version> represents the python version to used when installing KAT e.g. `/home/me/kat/lib/python3.6/site-packages`. Alternatively, you could install the kat python package into a python environment by changing into the `scripts` directory and typing `python setup.py install`.

CHAPTER 2

Using KAT

KAT is a C++ program containing a number of subtools which can be used in isolation or as part of a pipeline. Typing `kat --help` will show a list of the available subtools. Each subtool has its own help system which you can access by typing `kat <subtool> --help`.

2.1 HIST

Creates a histogram with the number of distinct k-mers having a given frequency, derived from the input. The input can take the form of one or more FastA or FastQ files, or a jellyfish hash. The last bucket in the histogram behaves as a catchall: it tallies all k-mers with a count greater or equal to the low end point of this bucket.

This tool is very similar to the `histo` tool in jellyfish itself. The primary difference being that the output contains metadata that make the histogram easier for the user to plot, and that this version is faster because we do not need to dump the hash to disk and read it back.

Basic usage:

```
kat hist [options] (<input>)+
```

Output:

Produces a histogram file and associated *Spectra hist* plot. The histogram file starts with a header describing settings used to generate the histogram. Each header line starts with a '#' character. The histogram that follows describes each K-mer frequency followed by the number of distinct K-mers found at that frequency separated by a space. Each frequency / count pair is line separated. For example:

```
# Title:27-mer spectra for: SRR519624_1.1M.fastq
# XLabel:27-mer frequency
# YLabel:# distinct 27-mers
# Kmer value:27
# Input 1:SRR519624_1.1M.fastq
###
1 47573743
2 4737789
3 732453
4 184505
5 100293
6 78699
```

(continues on next page)

(continued from previous page)

```
7 68553
8 59589
9 50926
...
```

Applications:

- Assess data quality: estimates of kmers deriving from errors; sequencing bias
- Determine completeness of sequencing
- Identify genomic properties: Heterozygosity, homozygosity, karyotype, repeat content.
- Limited contamination detection

2.2 GCP

This tool takes in either a single jellyfish hash or one or more FastA or FastQ input files and then counts the GC nucleotides for each distinct K-mer in the hash. For each GC count and K-mer coverage level, the number of distinct K-mers are counted and stored in a matrix. This matrix can be used in much that same way as a kmer spectra histogram, although it provides richer output by incorperating GC content into the picture. This helps to distinguish legitimate content from contamination, which often appear as separate spots at unexpected GC and coverage levels.

Basic usage:

```
kat gcp (<input>)+
```

Output:

Produces a matrix file and associated *Density* plot. The matrix file starts with a header describing settings used to generate the matrix. Each header line starts with a '#' character. The matrix that follows contains a space separated list of distinct k-mer counts for the GC-count indexed by the row. Each column index represents the K-mer Frequency. For example:

```
# Title:K-mer coverage vs GC count plot for: ERR409722_1.fastq ERR409722_2.fastq
# XLabel:K-mer multiplicity
# YLabel:GC count
# ZLabel:Distinct K-mers per bin
# Columns:1001
# Rows:27
# MaxVal:7705834
# Transpose:0
###
0 65392 10715 6038 4615 3769 3140 2690 2133 1748 1519 1370 1098 840 ...
0 189337 30772 20040 16630 15579 13673 12809 11890 10380 9605 8403 7370 6302 ...
0 428150 66753 41453 37478 34599 34622 34572 32740 31487 30356 28369 26880 ...
...
```

Applications:

- Assess data quality: estimates of kmers deriving from errors; sequencing bias
- Determine completeness of sequencing
- Identify genomic properties: Heterozygosity, homozygosity, karyotype, repeat content.
- Contamination detection

2.3 Comp

Compares jellyfish K-mer count hashes.

The most common use case for this tool is to compare two (or three) K-mer hashes. The typical use case for this tool is to compare K-mers from two K-mer hashes both representing K-mer counts for reads. However, it is also common to compare K-mers generated from reads to those generated from an assembly. If comparing K-mers from reads to K-mers from an assembly, the larger (most likely the read) K-mer hash should be provided first, then the assembly second. The third optional input acts as a filter, restricting the analysis to the K-mers present on that set. The manual contains more details on specific use cases.

Basic usage:

```
kat comp [options] <input_1> <input_2> [<input_3>]
```

Should the user wish to group multiple files to be concatenated into a single input group they may do so by surrounding the input group in single quotes. The following example groups the full input read set into the first input and compares against an assembly:

```
kat comp -t 8 -o pe_v_asm_test 'PE1.R1.fq PE1.R2.fq' asm.fa
```

. . . or more compactly:

```
kat comp -t 8 -o pe_v_asm_test PE1.R?.fq asm.fa
```

. . . or if the reads are gzipped:

```
kat comp -t 8 -o pe_v_asm_test PE1.R?.fq.gz asm.fa
```

If you are using multiple datasets for the input reads, instead of concatenating the files you can let KAT count over all the files in the group like this:

```
kat comp -t 8 -o pe_v_asm_test 'PE1.R?.fq.gz PE2.R?.fq.gz PE3.R?.fq.gz' asm.fa
```

Also KAT supports process substitution so if you wanted to use bzip2 compressed files you could do this:

```
kat comp -t 8 -o pe_v_asm_test <(bzip2 -dc 'PE1.R?.fq.bz2') asm.fa
```

KAT comp also allows 5' trimming of datasets (e.g. for barcode trimming of 10x data):

```
kat comp -t 8 -o pe_v_asm_test --d1_5ptrim 16,0,16,0 'PE1.R1.fq.gz PE1.R2.fq.gz␣
↪PE2.R1.fq.gz PE2.R2.fq.gz' asm.fa
```

Output:

Produces a matrix file and by default a *Spectra CN* plot, although can also produce a *Density* plot if requested. The matrix file is structured in a similar way to the GCP tool with a header describing settings used to generate the matrix. Each header line starts with a '#' character. The matrix that follows contains a space separated list of distinct k-mer counts for the frequency in each input file represented by the row and column index. For example:

```
# Title:K-mer comparison plot
# XLabel:K-mer multiplicity for: ERR409722_1.fastq
# YLabel:K-mer multiplicity for: ERR409722_2.fastq
# ZLabel:Distinct K-mers per bin
# Columns:1001
# Rows:1001
# MaxVal:57106148
# Transpose:1
###
0 57106148 2133673 428934 134189 45267 16399 6603 3374 2066 1371 930 752 490 ...
50919938 10364720 1613532 607932 239439 89985 36398 16589 8811 5469 3369 ...
```

(continues on next page)

(continued from previous page)

```
1990321 1605550 1061952 561999 271443 125163 61769 34379 22459 15647 11171 ...
...
```

Applications:

- Determine sequencing bias between left and right read pairs.
- Compare the kmer spectrum of input reads against an assembly to gauge assembly completeness.

2.4 SECT

Estimates coverage levels across sequences in the provided input sequence file. This tool will produce a FastA style representation of the input sequence file containing K-mer coverage counts mapped across each sequence separated by spaces. K-mer coverage is determined from the provided counts input file, which can be either one jellyfish hash, or one or more FastA / FastQ files. In addition, a file containing statistics about each target sequence is produced.

NOTE: K-mers containing any Ns derived from sequences in the sequence file not be included.

Basic usage:

```
kat sect [options] <sequence_file> (<input>)+
```

Output:

Produces a FastA-style representation of the K-mer frequency across each target sequence as well as a file describing statistics for each target sequence. The FastA-style output might look like this:

```
>Chr4
31 31 31 29 29 29 28 27 27 28 28 28 30 30 30 29 29 29 29 31 32 32 33 33 33 31 29␣
↪30 ...
```

With an associated tab separated stats file which looks like this:

```
seq_name median       mean      gc%  seq_length  kmers_in_seq  invalid_kmers  %_
↪invalid  non_zero_kmers  %_non_zero  %_non_zero_corrected
Chr4         26  362.45141  0.36204    18585056      18585036           3214    0.
↪01729        18549840    99.81065             99.82792
```

The column headers have the following meaning:

- seq_name - The name of the FastA/Q sequence
- median - The median K-mer coverage across the sequence
- mean - The mean K-mer coverage across the sequence
- gc% - The GC% of the sequence
- seq_length - The length of the sequence
- kmers_in_seq - The number of K-mers in the sequence. i.e. seq_length - K + 1
- invalid_kmers - The number of K-mers in the sequence that cannot be counted, most likely due to being non-canonical. i.e. non A,T,G,C
- %_invalid - The percentage of the sequence which contains invalid K-mers
- non_zero_kmers - The number of K-mers that have a coverage of 1 or greater
- %_non_zero - The percentage of the sequence which has a K-mer coverage greater than 1
- %_non_zero_corrected - The percentage of the sequence which has a K-mer coverage greater than 1 but ignoring any parts of the sequence represented by invalid K-mers.

Applications:

- Analyse K-mer coverage across assembled sequences
- Compare assemblies using K-mers, helpful to levels of contamination of a specific organism.
- Contamination detection - Compare K-mer spectrum against assembly providing average coverage and GC values for each contig, which can be 2D binned and plot as a heatmap

2.5 Filtering tools

KAT comes with two filtering tools allowing the user to slice and dice their data in a rapid and simple way.

2.5.1 K-mer filtering

This tool allows the user to produce K-mer hashes, within and outside user defined GC and k-mer coverage bounds. This is useful for isolating k-mers that could be attributable to contamination, or for contamination removal. Normally, the user would identify such regions using plots from the GCP tool.

Basic usage:

```
kat filter kmer [options] (<input>)+
```

Applications:

- Extracting k-mers with defined GC and coverage
- Contamination extraction (from k-mer hash)

2.5.2 Sequence filtering

The user loads a k-mer hash and then filters sequences (either in or out) depending on whether those sequences contain the k-mer or not. The user can also apply a threshold requiring X% of k-mers to be in the sequence before filtering is applied. The user can also use this tool for filtering paired end reads, and for subsampling.

Basic usage:

```
kat filter seq [options] --seq <seq_file> <k-mer_hash>
```

Applications:

- Contamination extraction from read file or assembly files, extraction of organelles, subsampling high_coverage regions

2.6 Plotting tools

KAT comes with a selection of plotting tools for representing and comparing K-mer spectra in various ways. All plotting tools come with the ability to manually modify axis, titles, limits, size, resolution, etc, although they will all try to pick intelligent defaults directly from the data provided.

2.6.1 Spectra hist

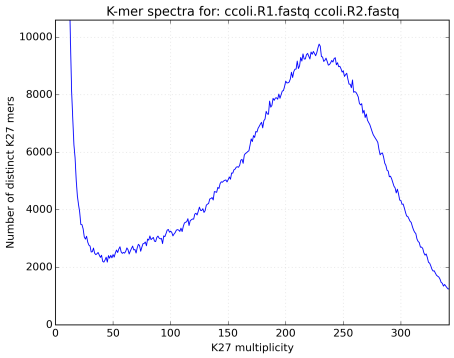
Visualises the K-mer spectra from `kat hist` or `jellyfish histo` output. This tool is designed to plot line graphs of one or more histograms. The idea is to be able to compare total K-mer counts between different datasets.

Basic usage:

```
kat plot spectra-hist <hist_file>
```

Applications:

- Basic K-mer spectra visualisation



2.6.2 Density

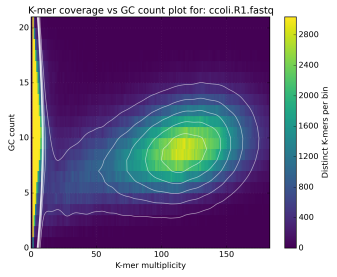
Creates a scatter plot, where the density or "heat" at each point represents the number of distinct K-mers at that point. Typically this is used to visualise a matrix produced by the `kat comp` tool to compare frequencies from two K-mer hashes produced by different NGS reads, or to visualise the GC vs K-mer matrices produced by the `kat gcp` tool.

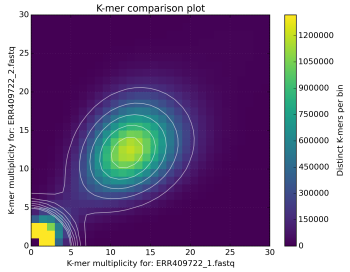
Basic usage:

```
kat plot density <matrix_file>
```

Applications:

- Visualise GC vs coverage matrices
- Visualise coverage vs coverage matrices





2.6.3 Profile

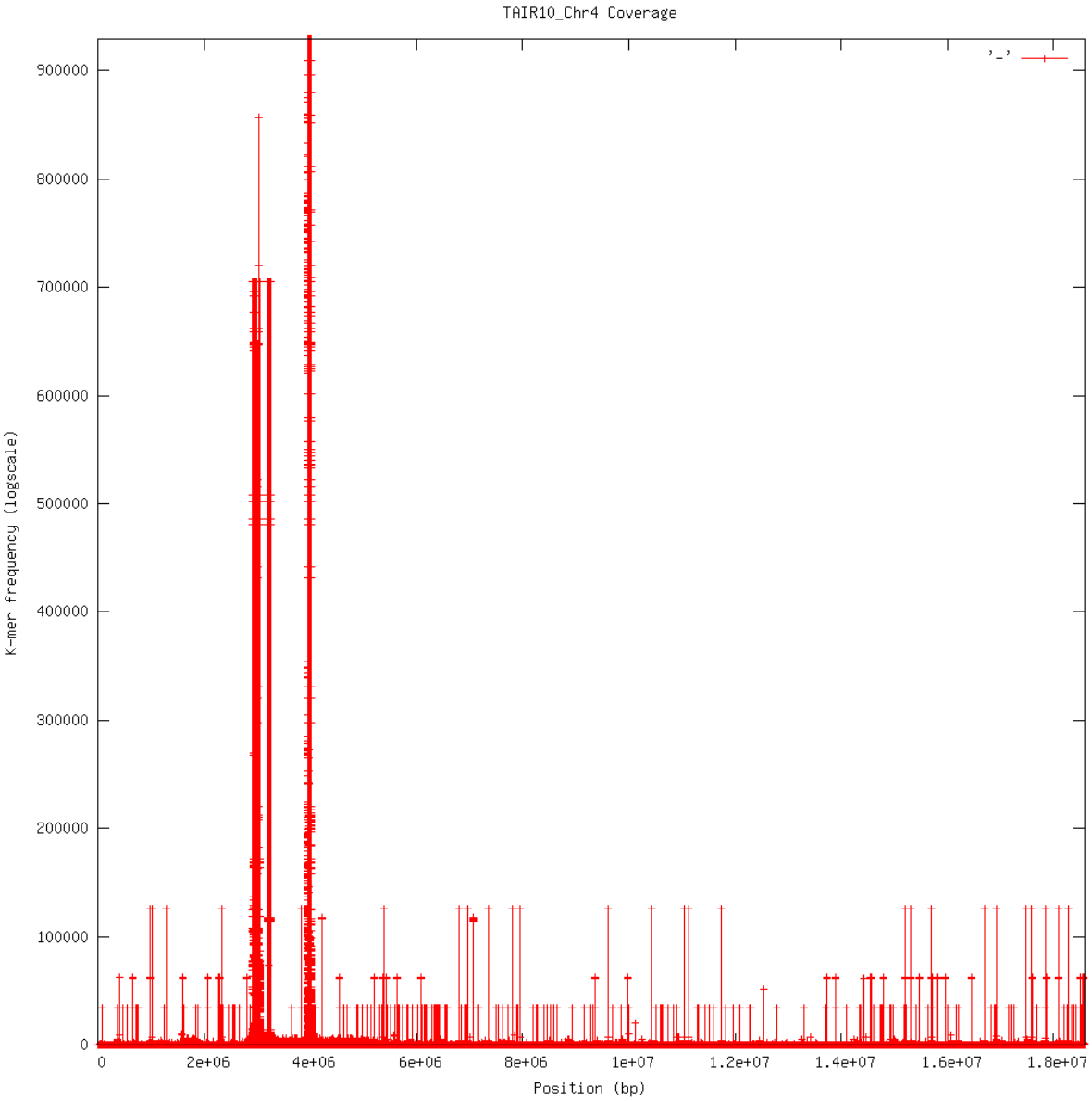
Shows K-mer coverage level across an sequence

Basic usage:

```
kat plot profile <sect_counts_file>
```

Applications:

- Visualise coverage (and optionally GC) levels across a sequence or set of sequences



2.6.4 Spectra CN

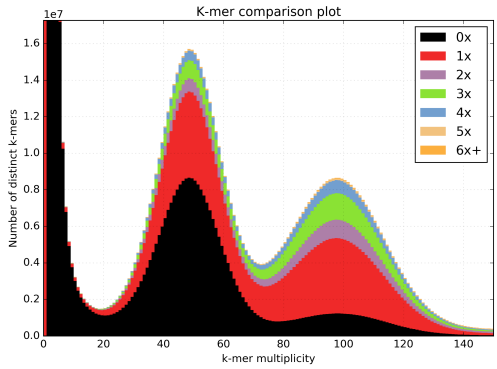
Shows K-mer duplication levels, which correspond to copy number variation within an assembly by comparing K-mers found in sequenced reads, to K-mers found in an assembly of those reads. Uses matrix output from the `kat comp` tool.

Basic usage:

```
kat plot spectra-cn <matrix_file>
```

Applications:

- Visualise the copy number spectra of WGS data compared against an assembly



2.6.5 Spectra MX

Produces K-mer spectras from rows or columns in a matrix generated by `kat comp`. This tool is designed to plot line graphs for one or more histograms, each histogram being represented by a single row or column in the matrix.

This tool also has a special mode for showing shared and exclusive content between two different samples. This mode takes the first row and column of the matrix representing content which is found exclusively in each sample. Two more lines are plotting, one which has each following row summed, and the other that has each following column summed. These two plots represent the shared content for each sample. This mode can be activated using the `--intersection` flag.

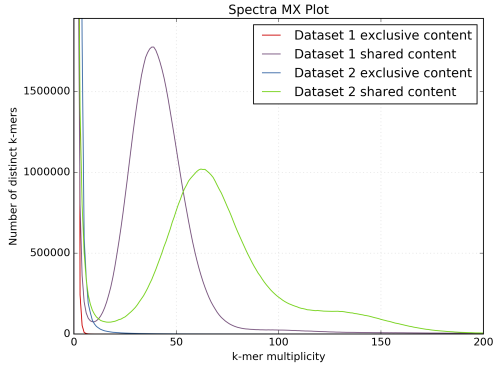
Alternatively, you can select specific rows and columns from the matrix using a comma separated list identified with the `--list` option. Each element in the list should start with either a 'c' or a 'r' indicating whether or not the column or row is requested. Then the element should contain a number indicating which column or row to select. For example: `--list c0,r1` will select column 0 and row 1. Note: spaces are not tolerated in this list.

Basic usage:

```
kat plot spectra-mx <matrix_file>
```

Applications:

- Visualising shared and exclusive content between two datasets
- RNAseq to WGS comparison
- Visualising k-mer spectra of arbitrary columns and rows from a matrix

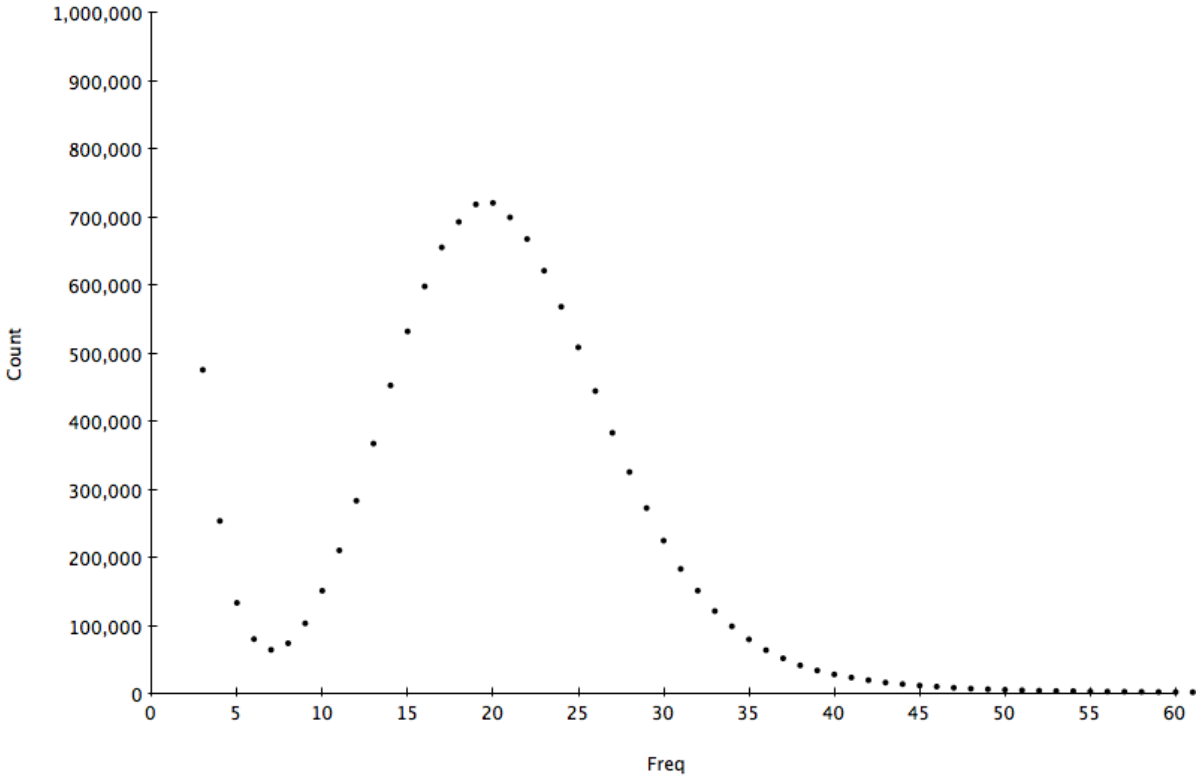


CHAPTER 3

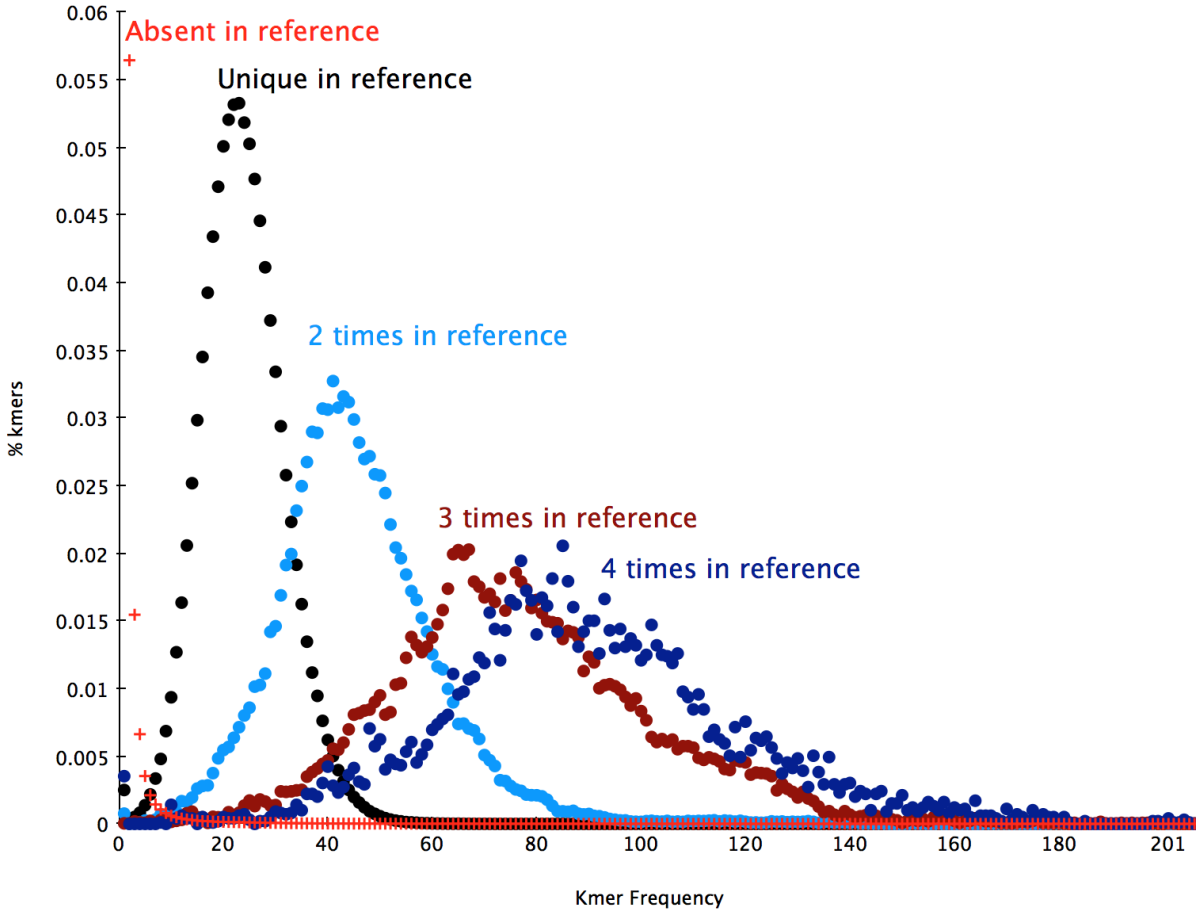
K-mer spectra

A K-mer spectra is a graphical representation of a dataset showing how many short fixed length words (k-mers) appear a certain number of times. The frequency of occurance is plotted on the x-axis and the number of k-mers on the y-axis. The k-mer spectra is composed of distributions representing groups of motifs at different frequencies in the sample, plus biases. Given not too many biases, the shape of the distributions provides a useful set of properties describing the biological sample, the sequencing process and the amount of useful data in the dataset.

A typical 31-mer spectrum of S.cerevisae S288C WGS dataset is shown in the following figure:



This is composed of an error component containing a huge amount of rare motifs at frequency < 7 arising from errors in the sequencing process, and a several other components as distributions with different modes according to how many times a motif appears on the genome (once, twice, three times etc.). The following plot shows the decomposition of this distribution into it's component distributions:


Absent in reference
Unique in reference
2 times in reference
3 times in reference
4 times in reference
% kmers
Kmer Frequency
0.06
0.055
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
0
20
40
60
80
100
120
140
160
180
201

CHAPTER 4

KAT Walkthrough

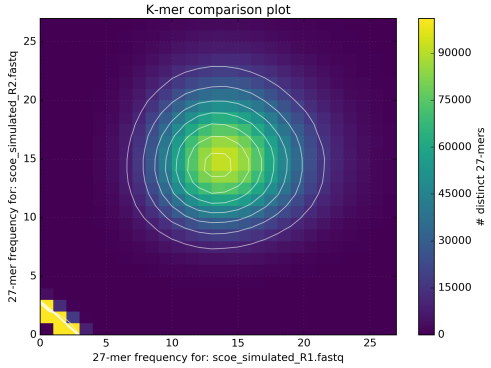
KAT is a multi-purpose toolkit and even we, the authors, have probably not fully mined all the possible scenarios in which the software can be applied. This section however lists some typical applications for which it has been used.

4.1 Comparing R1 v R2 in an Illumina PE dataset

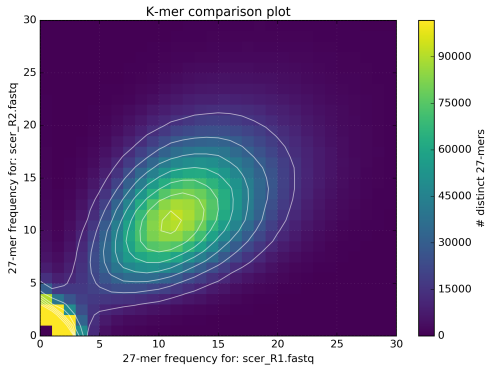
Comparing the k-mer spectra of read 1 to read 2 files of a paired-end (PE) sequencing run gives you several insights into properties such as shared errors, difference in quality, and provides a decent benchmark to compare two similar runs. Here's how to do it:

```
kat comp -t 16 -n -o <output_prefix> <R1.fastq> <R2.fastq>
```

Executing this command counts each fastq file into separate k-mer hashes, then creates a matrix of k-mer spectra frequencies in each dataset for each distinct k-mer. Finally, a density plot is made of the matrix. There are a number of interesting points in the output of the run. First, the some basic stats are produced on the standard output. Check properties like expected unique size, mean coverages for shared/unshared, or the distance measures between the spectra. These values are just indicative, but might point to disaster ahead. The density plot also provides a visual representation of how much shared errors are affecting your data. The following plot shows what you would expect to see for a completely unbiased homozygous PE library of S.coelicolor.

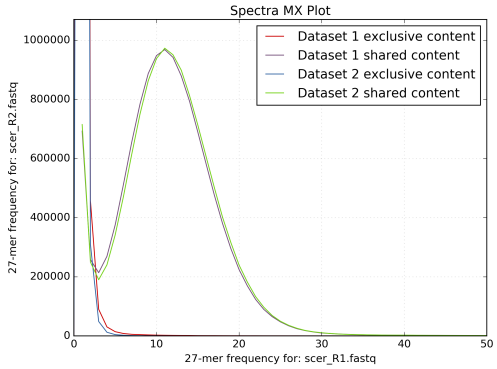


However in reality, various biases can interfere with sequencing experiments and we will probably end up with data which isn't quite as clean, such as that shown in the following plot from an Illumina run of S.cerevisiae S288C.



It's also useful to examine the shared and unique partitioned spectra between R1 and R2. This can be done by using the same matrix file. The plot below is from the same S.cerevisiae S288C dataset, generated using the following command line:

```
kat plot spectra-mx -i [options] -o <output_file> <matrix>
```



When generating plots, KAT uses input parameters supplied to the tool to generate a plot title and labels for the x- and y-axis. If you want to change these you can regenerate the plot from the matrix using the kat plot tool directly. For example, to regenerate the density plot from the S.coelicolor dataset above you can run the following command after running the kat comp command:

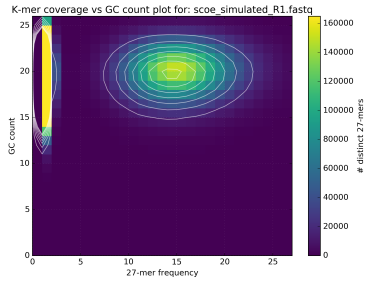
```
kat plot density -o <output_prefix> -t <title> -a <x-axis label> -b <y-axis label>
↪<matrix>
```

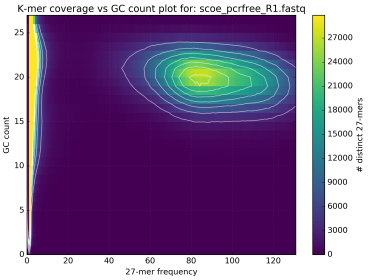
4.2 Detecting GC bias

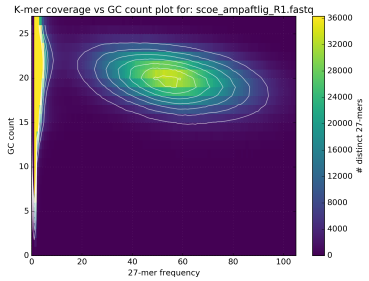
Using KAT it is possible to correlate biases due to GC content. KAT does this by combining k-mer frequency with the GC count for each distinct k-mer and representing the data in a matrix which can be plotted in a similar way to that discussed in the previous section. The command to produce a matrix of GC counts to k-mer frequency is as follows:

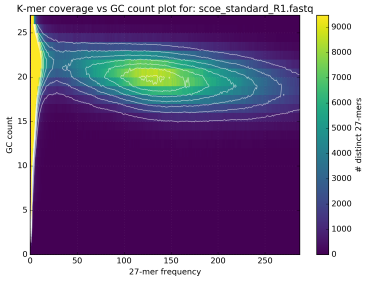
```
kat gcp -t 16 -o <output_prefix> (<fastq>)+
```

The following figure shows how GC bias varies depending on the protocols used in the sequencing experiments: 1) Simulated, 2) PCR-free, 3) PCR after adaptor ligation, 4) Standard protocol. In the simulated data there is a clean circle positioned at the expected GC levels for S.coelicolor, which has a GC rich genome. GC plots from the other experiments show a less distinct circle with the PCR-free protocol generating the truest reflection of the simulated dataset.









4.3 Checking library consistency

In most Whole Genome Shotgun (WGS) projects, you will use more than one library. The most simple case could be one PE library and one Long Mate Pair (LMP) library, but you could also have a dozen libraries of similar or completely different types, for example using different insert sizes or different library preparation protocols. It is good practice to check for obvious incongruence among libraries before trying to assemble them.

When you are sequencing the same genome, you are randomly sampling across it. Therefore you should have different spectrum originating from different experiments, but sampled from the same set. This means that if you decompose your spectra on the components being generated by single-copy elements, duplicated elements, triplicated elements, and so on, every motif belonging to a component distribution should belong to that component distribution across samples.

4.3.1 PE vs PE

Paired end sequencing constitutes the bulk of most current efforts on sequencing, and is used as a benchmark to sample the motifs on the genome. Since this data type is expected to have the more random distribution and even

coverage, it is a good place to start studying correlation between libraries.

The cleanest examples come from simulated data, where the correlation is virtually perfect as long as belonging to the same distribution, and shows almost no biases. Real data tends to show more correlation within the same distribution, especially in cases where strong biases are in play.

To compare two PE libraries run the following command:

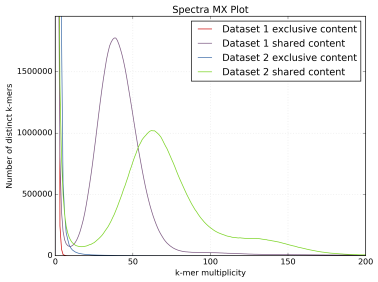
```
kat comp -n -t 16 -o pcrfree_vs_standard pcr_free 'pcr_free.R?.fastq' 'standard.R?.
↪fastq'
```

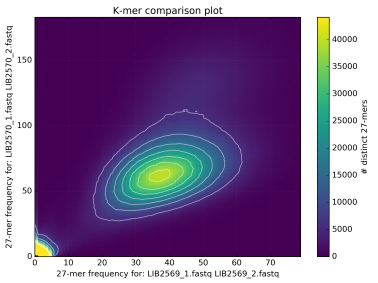
Note that the quotes around the inputs at the end of the command line allow you to group files together into a single input. Therefore all files matching "pcr_free.R?.fastq" are treated as the first input group, and all files matching "standard.R?.fastq" are treated as the second input group. K-mers are counted for each group separately. This saves the user wasting time and disk space concatenating PE files together prior to input into KAT.

The previous command produces only the density plot, so to generate the shared vs unique content plot also run:

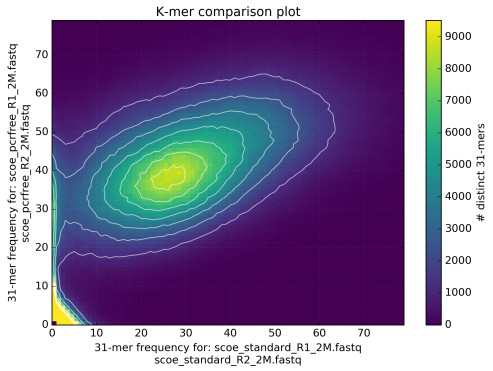
```
kat plot spectra-mx -i [options] -o pcrfree_vs_standard_shared.png <matrix_file>
```

The following plots compare two PE sequencing experiments in C.fraxinea, showing a large motif duplication in one of the experiments. This is obvious from the spectra-mx plot but not so clear in the density plot.

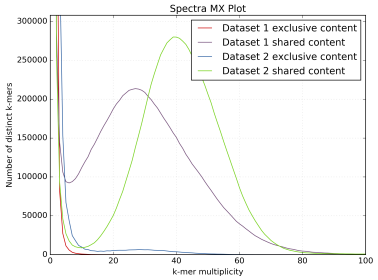


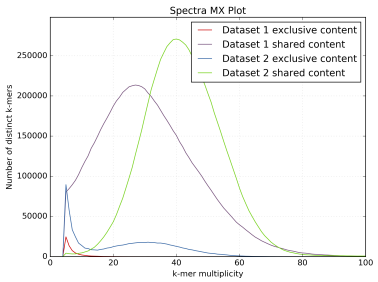


An interesting comparison to perform is between a PCR-free and a Standard protocol using a k-mer spectra density plot. Note that the coverage from the standard protocol is more variable than that generated from the PCR-free protocol. In addition, the blue region at (x=0 y=30) indicates kmers that are sampled by the PCR-free protocol but not the standard protocol. The coverage from the standard protocol is less than from the PCR-free protocol as less sequence was generated from this library.



The following shared content plots generated from the same comparison show the k-mer spectra split on shared and unique content. Note how content is "lost" on the standard protocol as soon as you ask for at least 5x kmer coverage. Although much of this is from the "error distribution" side (where the red and blue lines are truncated at K-mer multiplicity <5), you can also see that real content from the main frequency distribution is being lost by using that cutoff (the increased peak of the blue line around K-mer multiplicity = 30). This should make you think carefully about setting those low-coverage cutoffs again!

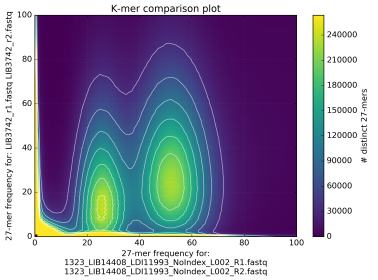


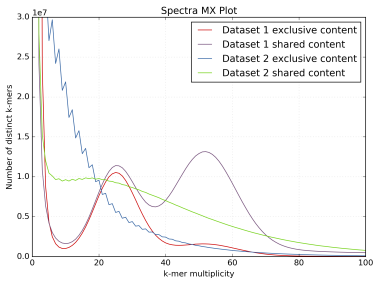


4.3.2 PE vs LMP

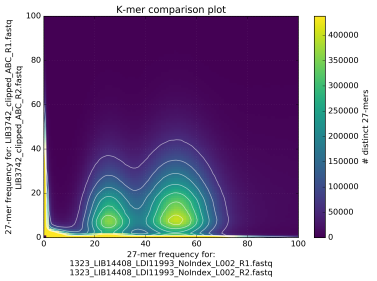
When using LMP data, in many cases the protocol used to prepare the sequencing library will impose considerable biases. It is good practice to check LMP reads against the PE reads for coherence. They have been prepared from the same genomic DNA so should have similar content. Over-representation and absence of motifs are important factors to check. The presence of motifs originating from adaptors (in fact mostly generated from their junction with genomic DNA) can also be spotted.

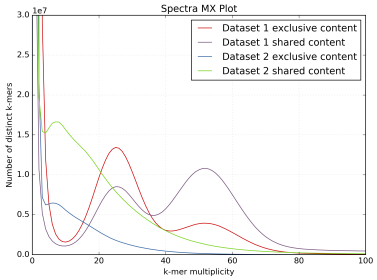
In the example shown below, a LMP run is compared to a PE run before processing according to the guidelines for the Nextera LMP protocol:





These plots show the same LMP run after processing:





While the motif presence and spectra of the LMP library are certainly better after processing (dataset 2 on the spectra-mx plot), there is content lost and the biases are clearly visible. You can spot representation bias on the density plot for both clusters on teh y-axis. Both clusters are too wide and have large "tails" going up. This is a typical signature for PCR-generated duplications in an early step in the protocol.

Especially interesting is the use of the shared and unique motifs to spot how well the LMP library covers the whole genome. It is usually accepted that for coverages higher than 10 the library should mostly cover the whole genome. If we look at the content "exclusive" to the PE library (the red line) as content not covered by the LMP library, it is obvious that processing the LMP removes a lot of content. While the spectra of the filtered LMP has better distribution, it is clear much content is not there. In this case, the library will not be very useful for scaffolding.

4.4 Contamination detection and extraction

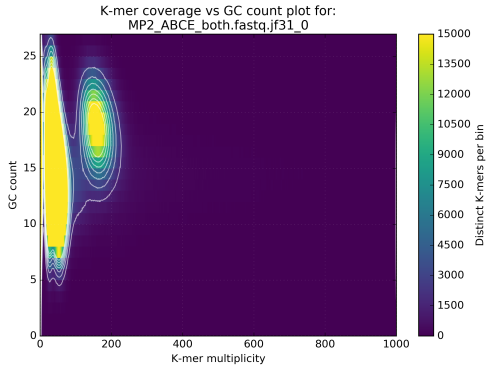
Breaking WGS data into k-mers provides a nice way of identifying contamination, organelles or otherwise unexpected content, in your reads or assemblies. This section will walk you through how you might be able to identify and extract contamination in your data.

4.4.1 In reads

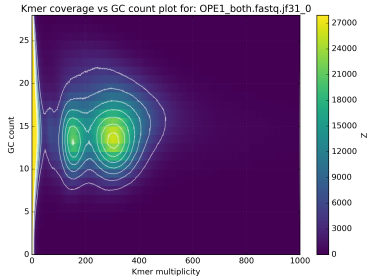
Detecting contamination in your WGS datasets are reliant on the contamination having differing levels of coverage and/or GC content from your target species. KAT can be used to identify this:

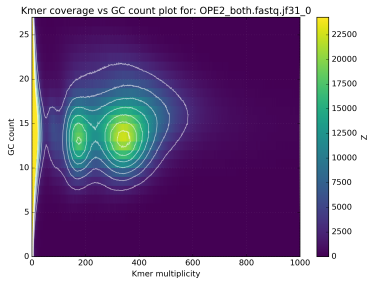
```
kat gcp [options] (<fastq>)+
```

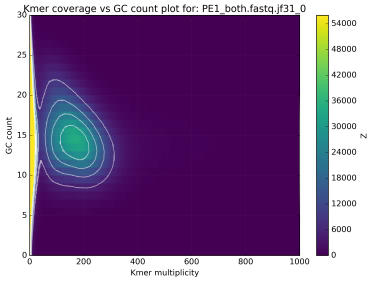
Running this tool will produce a matrix containing distinct k-mer counts at varying frequency and GC value. It will also produce a density plot, such as the one below that highlights error k-mers shown at very low coverage with a wide GC spread and genuine content between 10-100X with GC spread from 5-25. In this case we also have some unexpected content shown at approx 200X with GC from 15-25:



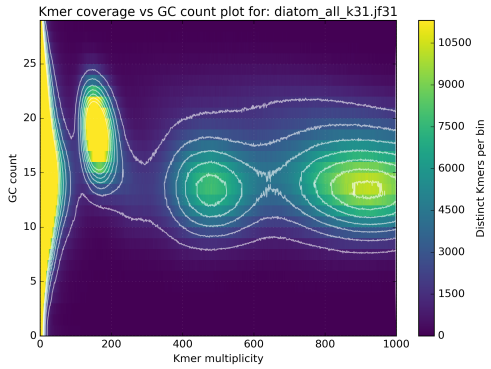
The high coverage hot-spot is already suspicious but it becomes even more so after looking at other WGS libraries of the same sample:







No other library contains such a hotspot at GC 15-25. After merging all libraries into one, the contaminant becomes obvious as the coverage has not altered, meaning that k-mers from this cluster were not found in the other libraries:



We can then use the filtering tools in KAT to extract k-mers inside, or outside defined coverage and GC limits. In this case we could take the original LMP library and run the following command:

```
kat filter kmer --low_count=100 --high_count=250 --low_gc=13 --high_gc=25 <path_to_
↪MP_lib>
```

This produces a k-mer hash containing only those k-mer found in the defined region. We can get the reads (or assembled contigs) associated with these k-mers by running the following command:

```
kat filter seq --threshold=0.5 --seq=<path_to_seq_file_to_filter> --seq2=<path_to_
↪seq_file_to_filter_2> <filtered_k-mer hash>
```

The example above assumes you want to filter a paired end library, although if you want to filter single end data or and assembly you can do this by simply dropping the `--seq2` option.

BLASTing some of the sequences removed by the filtering might then identify the contaminant.

You can also use this tool for subsampling the extracted data. This can be useful for reducing expression of highly expressed reads. To do this add the `--frequency` option and set a threshold indicating how many of the reads to keep: 1.0 implies keep all, 0.0 means discard all, 0.5 would imply to keep half of the sequences.

4.4.2 In assemblies

Detecting contaminants in assemblies involves a similar process to that described in the previous section. It involves marking contigs in an assembly with their average k-mer coverage and GC%.

To obtain the average coverage and GC% scores for each contig use the following command:

```
kat sect [options] <assembly> (<fastq>)+
```

By extracting the median coverage and gc% columns from the stats file it is possible to create a scatter plot which can be used in a similar way to that described in the previous section.

A second use case assumes you already know the contaminant genome and have access to the reference assembly of that contaminant. In this case you can directly inspect your assembly for signs of the contaminant using the following command:

```
kat sect [options] <assembly> <contaminant_genome>
```

This counts k-mers in the contaminant genome and applies them to the sequences in your assembly. By reverse sorting the stats file produced by the "%_non_zero_corrected" column you can identify contigs belonging to the contaminant. Normally, assuming the contaminant is the exact same species as that found in your assembly you expect to see very high percentage scores (>90%). Moderate scores (20-80%) might indicate either some shared content or chimeric sequences and should be investigated more thoroughly.

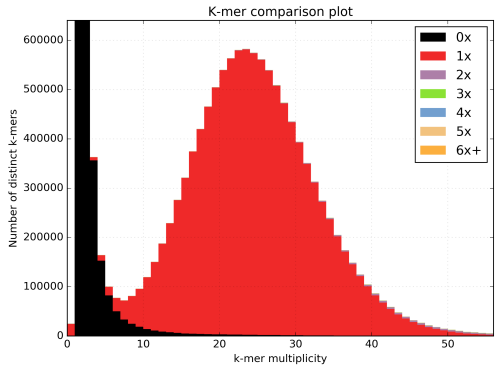
4.5 Genome assembly analysis using k-mer spectra

One of the most frequently used tools in KAT are the so called "assembly spectra copy number plots" or spectra-cn. We use these as a fast first analysis to check assembly coherence against the content within reads that were used to produce the assembly. Basically we represent how many elements of each frequency on the read's spectrum ended up not included in the assembly, included once, included twice etc.

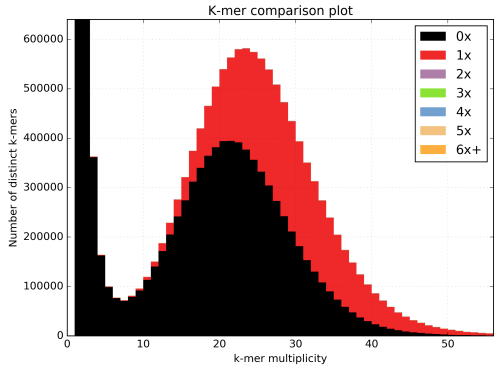
4.5.1 Homozygous genomes

As a simple example, we can look at how a plot for S.cerevesiae S288C would look if we are able to perfectly reconstruct the reference assembly:

```
kat comp -t 16 -o pe_vs_assembly 'PE.R?.fastq' assembly.fa
```



The errors are absent on the assembly, the main unique content is all there, exactly once, and all the other distributions are perfectly in place. But from the same sequencing, by choosing a wrong k-value during assembly (too small in this case), we can end up with something more interesting.

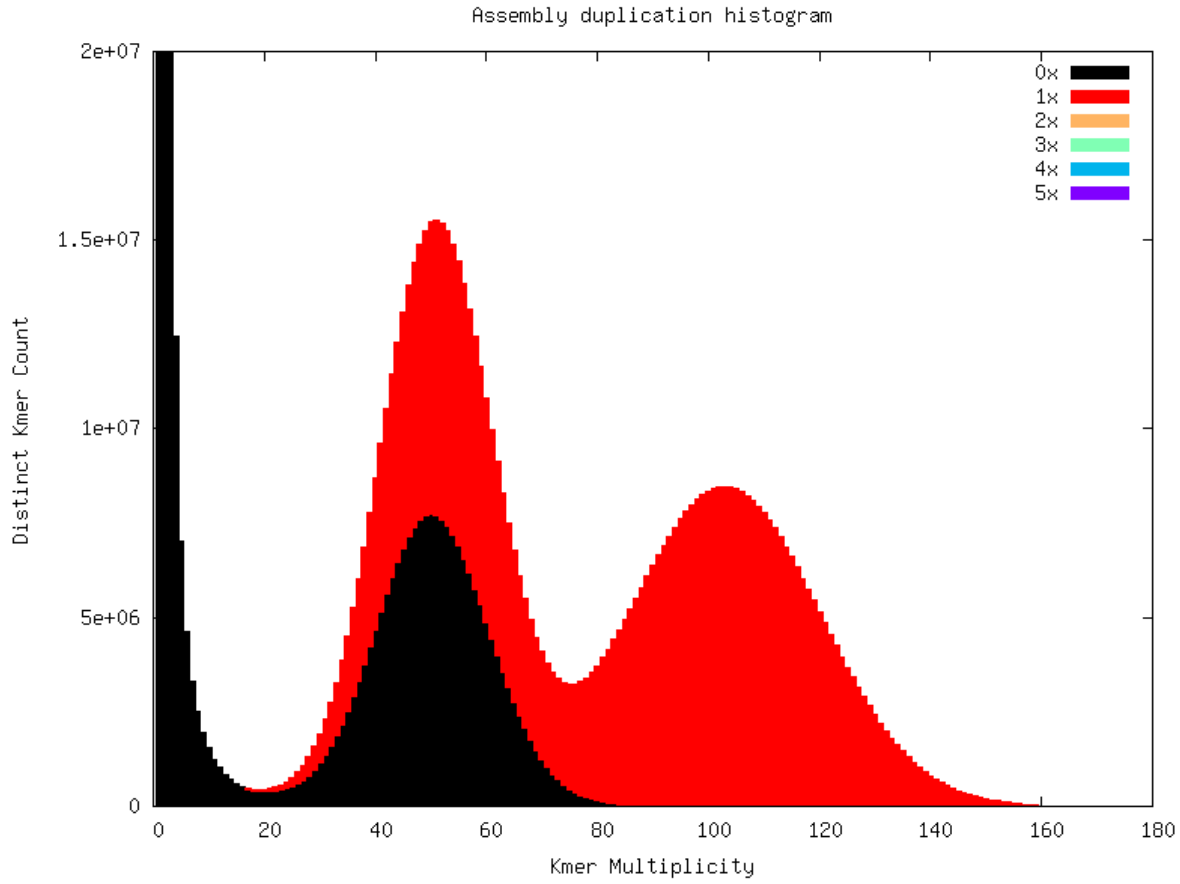


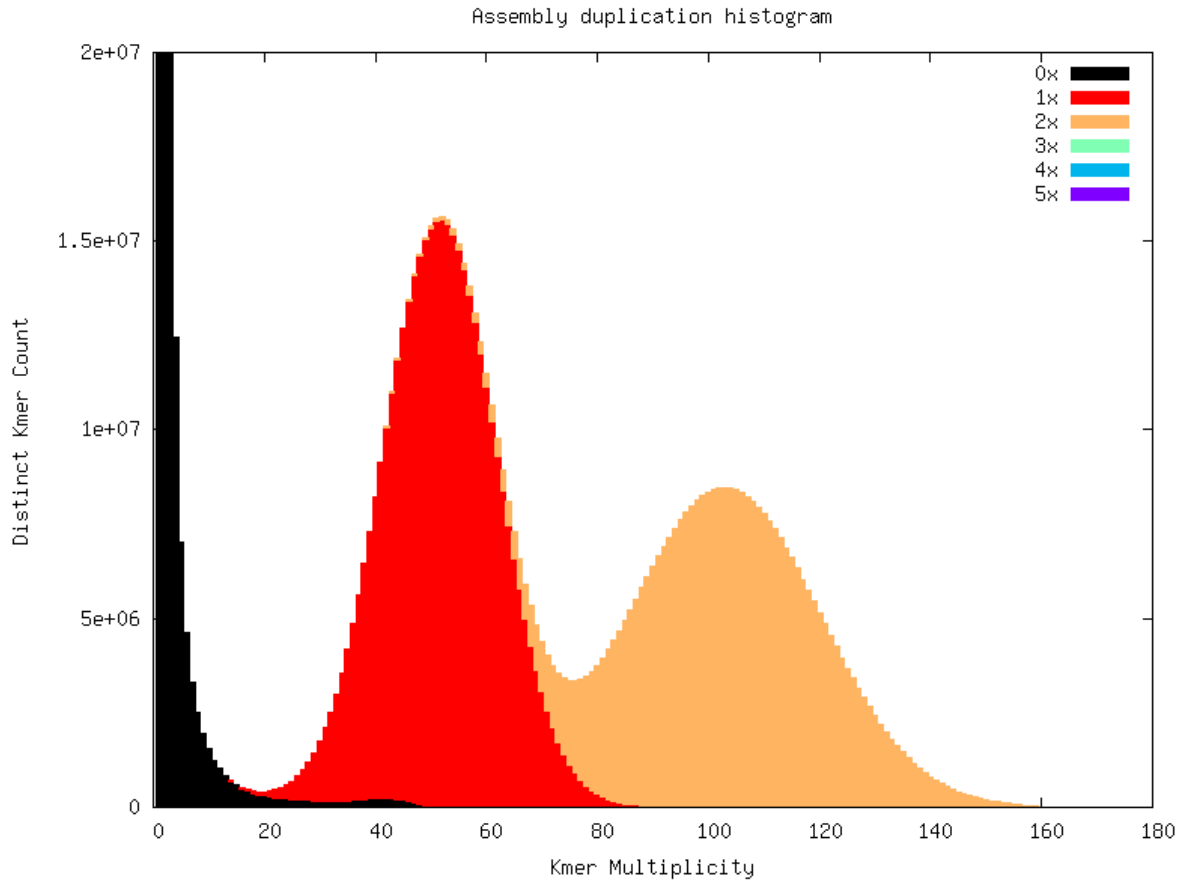
Now, in addition to the absent errors, we have a lot of missing content from the assembly.

Sometimes when we generate an assembly we want to remove short contings from the final assembly as these contigs are often not useful in downstream analysis. It is common to remove contigs shorter than 200bp, 500bp or 1Kbp but it can be a problem deciding which cutoff to use as you don't want to remove useful content from the assembly. The spectra-cn plot is useful here as you can check assembly files (with no cutoff, 200 bp cutoff, 500bp cutoff etc.) using kat comp to quantify the content you are removing using that cutoff. Missing content is evident as a black peak below the main red peak and will increase in height as you remove more content. The choice of cutoff is a trade-off between reducing the number of contigs in the assembly and keeping as much content as possible.

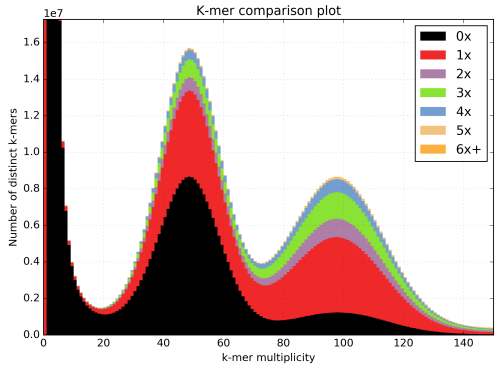
4.5.2 Heterozygous genomes

Heterozygous genomes produce more interesting and complex plots, since the k-mer spectra clearly shows different distributions for both the heterozygous and homozygous content. The following plots show the two extremes of how a heterozygous assembly could look. The hererozygous content is represented by the first peak at x=50 and the homozygous content in the second peak at x=100. In the first plot we have a single haplotype mosaic, where the bubbles in the graph are collapsed and each heterozygous region is represented once in the assembly. This is what we typically would expect to get out of a perfect assembly. The lost content (the black peak) represents the half of the heterozygous content that is lost when bubbles are collapsed. In the second case, haplotypes are separated by duplicating all the homozygous regions and allowing us to fully capture the heterozygous content. We don't typically, aim for the second scenario when assembling genomes.





Interestingly, most assemblies don't look like either case above but show duplications, inclusion of extra variation, etc:



4.6 Distribution decomposition analysis

It's useful to be able to fit distributions to each peak in a k-mer histogram, spectra-cn matrix in order to work out how many distinct k-mers can be associated with those distributions. By counting k-mers in this way we can make predictions around genome size, heterozygous rates (if diploid) and assembly completeness. To this end we bundle a script with kat called kat_distanalysis.py. It takes in either a spectra-cn matrix file, or kat histogram file as input, then proceeds to identify peaks and fit distributions to each one. In the case of spectra-cn matrix files it also identifies peaks for each copy number for an assembly. Alternatively, for matrix files generated by KAT GCP, it will also identify peaks associated with GC content. Output from this tool consists of stdout logging as well as structured JSON output. In addition, plots of the fitted distributions can be requested using the *–plot* option.

The user can help to get correct predictions out of the tool by providing an approximate frequency for the homozy-

gous part of the distribution. By default, this is assumed to be the last peak. For example, this command:

```
kat_distanalysis.py --plot spectra-cn.mx
```

might produce the following output for this tetraploid genome:

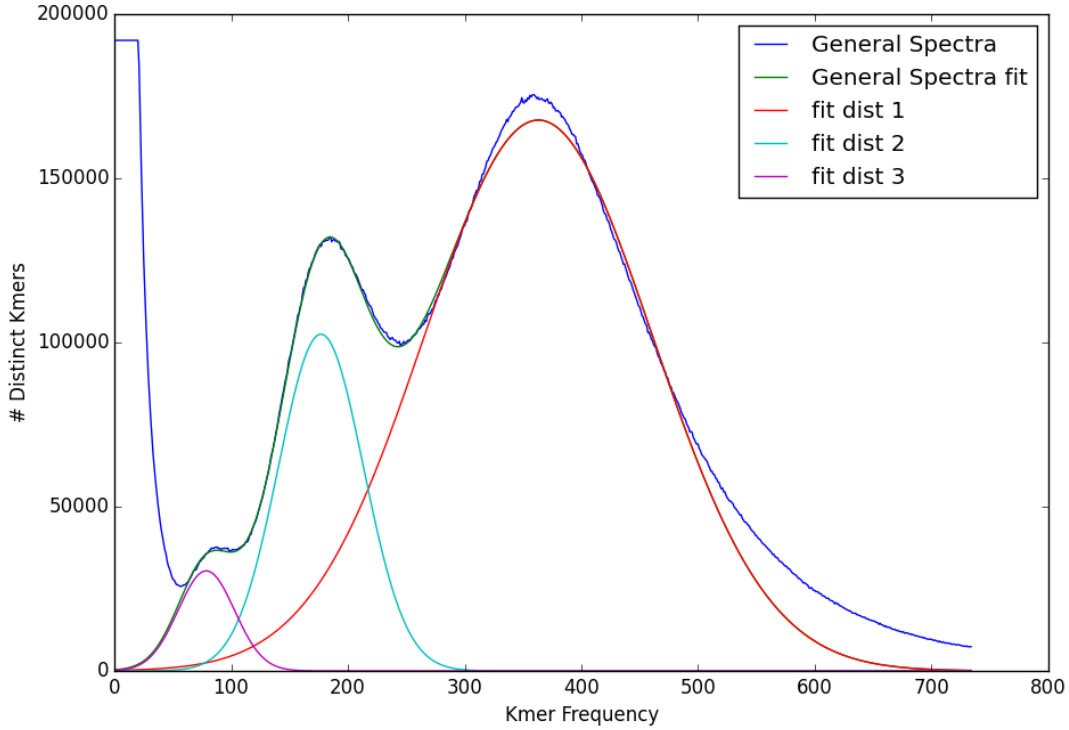
```
Main spectra statistics
-----------------------
K-value used: 27
Peaks in analysis: 3
Index   Freq    Max     Volume
1       79      35596   1844622
2       177     132229  9354614
3       364     175454  41202694
Mean k-mer frequency: 320x
Homozygous peak index: 3
Estimated genome size: 48.05 Mbp
Estimated heterozygous rate: 0.86 %
Estimated assembly completeness: 95.73%

Breakdown of copy number composition for each peak
--------------------------------------------------

---- Report for f=364.209 (total elements 36483749)----
 0x: No significant content
 1x: 100.00% (36483749 elements at f=367.62)
 2x: No significant content
 3x: No significant content

---- Report for f=177.838 (total elements 9214569)----
 0x: 47.33% (4361617 elements at f=183.02)
 1x: 52.67% (4852951 elements at f=176.43)
 2x: No significant content
 3x: No significant content

---- Report for f=79.618 (total elements 858966)----
 0x: 100.00% (858966 elements at f=77.18)
 1x: No significant content
 2x: No significant content
 3x: No significant content
```



As of KAT V2.4.0, this script is executed as a post-processing step to most KAT tools.

4.7 Finding repetitive regions in assemblies

Sometimes it's useful to identify regions in an assembly that are repetitive. This can easily be done with the following command:

```
kat sect -E -F [options] <assembly_file> <assembly_file>
```

This counts k-mers in the assembly and then marks up the sequences in the assembly with k-mer counts at each position. Regions that have a count of 1 are extracted into a new FastA file containing non-repetitive content and regions that have a count of 2-20 (maximum threshold can be adjusted) are extracted to FastA file containing the repetitive content.

CHAPTER 5

Frequently Asked Questions

5.1 Can KAT handle compressed sequence files?

Yes, as of V2.4.0, KAT has native support for gzip decompression, so just treat gzipped files as regular uncompressed fastq or fasta files.

If you wish to decompress other files such as bzip (or if you are using a pre V2.4.0 KAT), then this is supported via named pipes. Anonymous named pipes (process substitution) is also supported.

For example, say we wanted to run `kat hist` using bz2 paired end dataset, we can use a named pipe to do this as follows:

```
mkfifo pe_dataset.fq && bzip2 -d -c pe_dataset_?.fq.bz2 > pe_dataset.fq &
kat hist -o pe_dataset.hist pe_dataset.fq
```

Where `pe_dataset_?.fq.bz2`, represents `pe_dataset_1.fq.bz2` and `pe_dataset_2.fq.bz2`.

For those unfamiliar with named pipes, the first line will create an empty file in your working directory called pe_dataset.fq and then specifies that anything consuming from the named pipe will take data that has been gunzipped first. To be clear this means you do not have to decompress the gzipped files to disk, this happens on the fly as consumed by KAT.

Alternatively, using process substitution we could write the previous example more concisely in a single line like this:

```
kat hist -o oe_dataset.hist <(bzip2 -d -c pe_dataset_?.fq.bz2)
```

As a more complex example, the KAT comp tool can be driven in spectra-cn mode using both compressed paired end reads and a compressed assembly as follows:

```
kat comp -o oe_spectra_cn <(bzip2 -d -c pe_dataset_?.fq.bz2) <(bzip2 -d -c asm.fa.
↪bz2)
```

Thanks to John Davey and Torsten Seeman for suggesting this.

5.2 Why is jellyfish bundled with KAT?

We require a stable interface to the k-mer hash arrays produced by jellyfish hence, we are reliant on a particular version of jellyfish to guarantee that KAT works correctly. Instead of potentially requiring the user to install multiple jellyfish instances on their machine, we bundle our own version, with all jellyfish binaries prefixed with *kat_* in order to avoid any naming clashes with official jellyfish releases. We have also made several modifications to jellyfish which make it more suitable to processing via KAT.

5.3 Where's the distributable tarball for post-V2.4.0 releases?

As of V2.4.0, we no longer support installation via tarball. We did this in order to ensure boost is built alongside KAT, and this just didn't fit well into the `make dist` mechanism. Please follow the new installation instructions and download KAT via `git clone`.

5.4 Should I dump jellyfish hashes to disk?

Most KAT tools have an option `-d` to write out the jellyfish hash to disk. While it may seem logical to do this if you need to reprocess the data, in normal usage I do not recommend it. Reading in K-mers straight from fastq or fasta, in most circumstances, will be faster than loading a jellyfish hash directly, particularly when using 4 or more threads (and especially, if the input files are gzipped). Also storing jellyfish hashes can consume a large amount of storage space.

CHAPTER 6

System requirements

KAT supports Unix, linux or Mac systems. Windows, with something like cygwin, may work but hasn't been tested. A minimum of 8GB RAM, which will enable you to process small - medium sized datasets. Large datasets will require more RAM (potentially a lot more), the actual amount of memory required depends on the size of the genome's to be processed, the k-mer size selected and the size of your datasets.

CHAPTER 7

Citing

If you use KAT in your work and wish to cite us please use the following citation:

Daniel Mapleson, Gonzalo Garcia Accinelli, George Kettleborough, Jonathan Wright, and Bernardo J. Clavijo. **KAT: A K-mer Analysis Toolkit to quality control NGS datasets and genome assemblies.** Bioinformatics, 2016. doi: 10.1093/bioinformatics/btw663

CHAPTER 8

Issues

Should you discover any issues with KAT, or wish to request a new feature please raise a ticket here. Alternatively, contact Daniel Mapleson at: d.mapleson@gmail.com; or Bernardo Clavijo at: bernardo.clavijo@earlham.ac.uk. However, please consult the Frequently Asked Questions page first in case your question is already answered there.

CHAPTER 9

Availability and License

Open source code available on github: https://github.com/TGAC/KAT.git

This documentation is hosted publicablly on read the docs: https://kat.readthedocs.org/en/latest/

KAT is available under GNU GLP V3

CHAPTER 10

Acknowledgements

We owe a big acknowledgment to all TGAC staff that has been bored eternally with k-mers, you have all been incredible patient and supportive with us.

Thanks to Mario Caccamo, Sarah Ayling, Federica Di Palma and David Swarbreck for all the support, feedback and encouragement.

Thanks to Richard Leggett, Daniel Zerbino and Zamin Iqbal for all the interesting discussions, comments and input.

Thanks to Dan Sargent for the use of his P.micrantha datasets for tests, and their inclusion as figures on this document.

Thanks to all the KAT early adopters users who have provided invaluable feedback on the tool in its early stages: Paul Bailey, Jose De Vega, Rocio Enriquez-Gasca, Marco Ferrarini and Dharanya Sampath. And more recently, those from further afield who have contributed on github.

A big thanks to the author of jellyfish, Guillaume Marcais. Jellyfish is fantastic piece of software and is critical to enabling KAT to do what it does in an efficient and timely fashion.

Last but not least a very special thanks to the Lab guys on their white coats, trying to make sense of all our comments, giving us better data each day and trying to get into our heads all the complex explanations for the biases and extra variability we were finding day after day.

CHAPTER 11

Credits

- Daniel Mapleson (The software architect and developer)
- Gonzalo Garcia (KAT superuser and primary tester)
- George Kettleborough (For the recent python plotting functionality)
- Jon Wright (KAT superuser and documentation writer)
- Bernardo Clavijo (KAT's godfather, evangelist and all-round k-mer guru)