The Karyon v1.0 manual
By Miguel Naranjo-Ortiz
March 3rd 2021

About Karyon

Even for organisms with small compact genomes, the production of a high quality reference genome is often an arduous prospect. Many biological circumstances can act as obstacles for your typical genome assembly project, some of which might be previously unknown. The goal of Karyon is to provide an easy to use pipeline for the generation of a reference assembly and a series of analyses designed to identify sources of conflict.

1. Quick start

Before starting, you need to decide whether to use standard installation or to install Karyon in a Docker container.

2. Installation (Standard)

For installing Karyon from GitHub:

```
git clone https://github.com/Gabaldonlab/karyon.git

cd karyon/scripts/

sudo bash scripts/install.sh
```

This will create everything you need to run Karyon in your system.

3. Light installation (Standard)

Users who are interested in only using the plotting part of the pipeline might want to use the script lightinstall.sh. This will install the required dependencies for running the script allplots.py, but will skip most of the required programs for running the main karyon script. This version is faster to install and will require much less disk space.

```
git clone https://github.com/Gabaldonlab/karyon.git

cd karyon/scripts/

sudo bash scripts/lightinstall.sh
```

4. Installation (Docker)

Prerequisites

In order to run this container you'll need docker installed.

- [Windows]
- [OS X]
- [Linux]

Download stable container

Pull `gabaldonlab/karyon` from the Docker repository:

```
docker pull gabaldonlab/karyon
```

Build your own container

Or build `gabaldonlab/karyon` from source:

```
git clone https://github.com/Gabaldonlab/karyon_docker.git
```

```
cd karyon_docker

docker build -t cgenomics/karyonpip:1.0 .
```

Running the image

Every time you run a Docker container, it uses the Karyon Docker image to remake it. When the container is closed, everything in it goes away! Also, for every one of those containers you need to name it with an alias and include a volume to work with.

```
docker run -dit --name=your-alias -v
/your/karyon/location:/root/src/karyon/shared --rm
cgenomics/karyonpip:1.0
```

Launch Karyon script

```
docker exec -w /your/karyon/location/ karyon python scripts/karyon.py
-l shared/your-data-file-one shared/your-data-file-two... -d
shared/output -T
```

Ports

This container exposes the following ports:

Port	Protocol	Service
8080	TCP	http

5. Basic usage

Once everything has been properly installed, you can run Karyon. In its most simple form, Karyon will take a set of Illumina libraries and a name for the output directory where it will generate everything. Be aware that Karyon will create several files that are typically quite large. Be sure you have plenty of space in your hard drive. Final size will depend on the amount and size of input libraries, but 50Gb of free disk space should be more than enough for most cases.

Let us assume we are in a directory that contains the sequencing libraries L1_1.fastq, L1_2.fastq, L2_1.fastq and L2_2.fastq; and we want Karyon to generate the results in a directory called “karyon_results”. The command for that is:

```
python3 karyon.py -l L1_1.fastq L1_2.fastq L2_1.fastq L2_2.fastq -d
karyon_results
```

By default, it will create a random string of characters to use as a prefix for all the output files. If we want to set a name, we can do it with the flag `-o` or `--output_name`. For example, if we want to give the prefix “example”, we can do it with:

```
python3 karyon.py -l L1_1.fastq L1_2.fastq L2_1.fastq L2_2.fastq -d karyon_results -o example
```

6. The config file

The installation would have generated a file in the scripts directory called configuration.txt. This is read by Karyon to figure out where the dependencies are located and the system, as well as introducing non-default parameters. The file contains several lines that start with “#”, which are ignored.

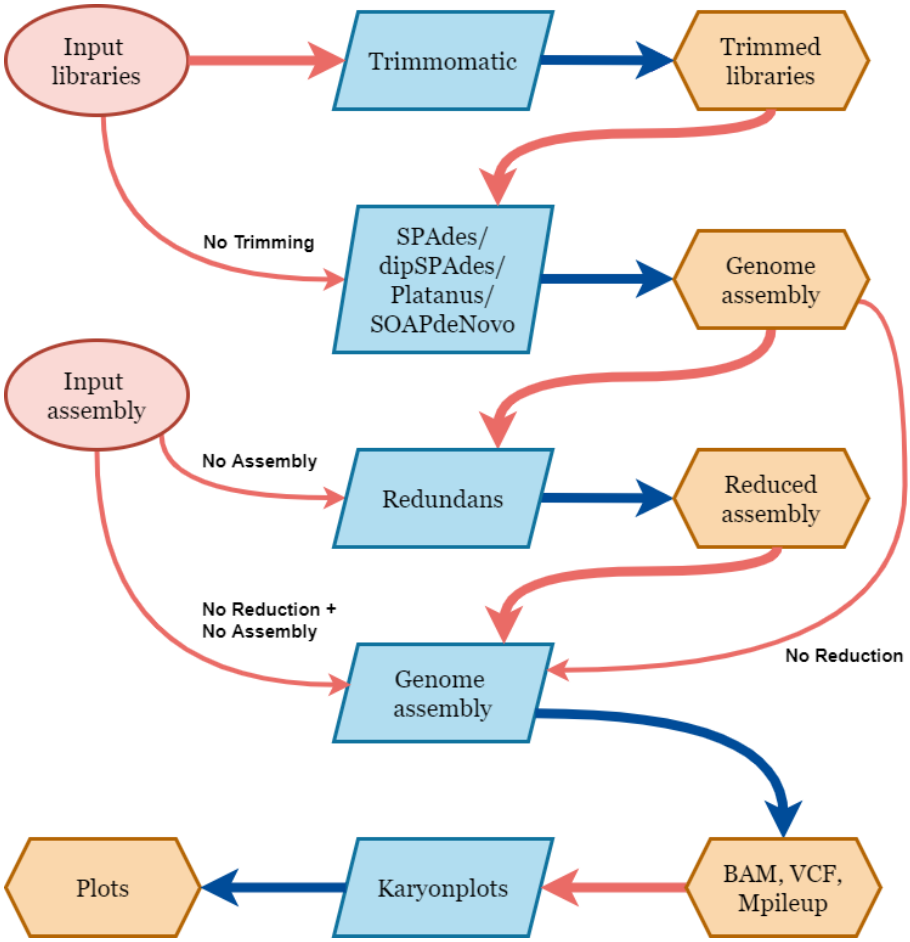
For each of the programs used by Karyon we have three lines:

- The name of the program starts with “+”
- The location of the program starts with “@”. An empty line implies that the program can be accessed directly from the environment.
- Any additional flag can be introduced with a line starting with “>”. Each additional flag can be inputted with an additional line.

You can input a different config file with the flag `-c` or `–configuration`.

7. The Karyon pipeline

The Karyon pipeline consists on the following steps:



- **Trimming**: Remove adapters and low quality positions from Illumina reads.
- **Genome assembly**: Generate a *de novo* assembly using the cleaned libraries.
- **Reduction**: Karyon uses Redundans to improve genome assembly quality by collapsing contigs with high similarity. This produces an assembly with reduced heterozygosity that will be used as reference for variant calling.
- **Variant calling**: One of the original libraries will be mapped against the reference in a standard variant calling pipeline.
- **Plot generation**: Using the results of the previous steps, Karyon will generate a series of plots that can help to understand the genomic architecture of the genome.

6.a. Trimming

By default, Karyon will apply Trimmomatic (http://www.usadellab.org/cms/?page=trimmomatic) to the libraries selected as input. It might happen that our libraries are already trimmed, and we want to skip that step. This can be done with the flag `-T` or `--no_trimming`.

```
python3 karyon.py -l L1_1.fastq L1_2.fastq L2_1.fastq L2_2.fastq -d karyon_results -T
```

6.b. Genome assembly

Karyon is able to use several popular genome assemblers for the creation of the initial assembly. By default it will use dipSPAdes, which is an implementation of the SPAdes genome assembler for heterozygous diploid genomes (https://github.com/ablab/spades/tree/spades_3.9.0). You can run Karyon with the standard SPAdes, Platanus (http://platanus.bio.titech.ac.jp/platanus-assembler) or SOAPdeNovo2 (https://github.com/aquaskyline/SOAPdenovo2). You can specify which program to use with the flag `-g` or `–genome_assembler`. Accepted values are 'dipspades', 'dipSPAdes', 'spades', 'SPAdes', 'platanus', 'Platanus', 'soapdenovo' and 'SOAPdenovo'.

For example, if we want to run the previous example using Platanus as our genome assembler, we can use the following command:

```
python3 karyon.py -l L1_1.fastq L1_2.fastq L2_1.fastq L2_2.fastq -d
karyon_results -g platanus
```

Alternatively, we might already have a genome assembly that we want to use. In that case, we can provide the flag `-A` or `–no_assembly`. This will tell Karyon not to run the genome assembly, and requires a fasta file that will be used for subsequent steps. For the previous example, imagine that we already have a genome assembly in a file called example.fasta. We can use it instead of running the assembly with the following command:

```
python3 karyon.py -l L1_1.fastq L1_2.fastq L2_1.fastq L2_2.fastq -d
karyon_results -A example.fasta
```

6.c. Reduction

Karyon uses Redundans (https://github.com/lpryszcz/redundans) to generate a reduced assembly that will be used as reference in the variant calling protocol. You can skip this step with the flag `-R` or `--no_reduction`. If you skip it, the reference genome will be the result of the genome assembly step. If the genome assembly step is also omitted, the reference will be the fasta file provided as input.

6.d. Variant calling

Karyon will use the assembly generated or provided by previous steps as reference for a variant calling pipeline using BWA-MEM (https://github.com/lh3/bwa), GATK (https://github.com/broadinstitute/gatk) and Bedtools (https://github.com/arq5x/bedtools2). This part of the pipeline can be skipped with the flag `-V` or `--no_var_call`. However, doing so will also skip plot generation, as the plots require the results of this step.

In order to save time, Karyon will only use one library for the variant calling protocol, which is internally referred as the “favourite” library. The default behavior is that the favourite will be to select the library with higher depth of sequencing. You can manually select which library to use

as favourite with the flag `-F` or `--favourite`. Let's imagine that we want to use L2_1.fastq as favourite in the previous example:

```
python3 karyon.py -l L1_1.fastq L1_2.fastq L2_1.fastq L2_2.fastq -d karyon_results -F L2_1.fastq
```

6.e. Plot generation

Karyon will automatically use the results of the variant calling pipeline to create the plots. Karyon allows to configure some of the parameters used for plotting:

- Window size (`-w` or `--wsize`): Defined the length of the genome pieces used for the plots. Default is 1000. Low values will generate very noisy results, while large values will produce plots with very low resolution.
- Maximum number of scaffolds to plot (`-x` or `–max_scaf2plot`). Sets a limit to the number of scaffolds to be represented in several plots. Default is set to 20.
- Minimum scaffold length (`-s` or `--scafminsize`). Scaffolds with a length below a threshold will be ignored. If not defined, it will apply no filter.
- Maximum scaffold length (`-S` or `--scafmaxsize`). Scaffolds with a length above a threshold will be ignored. If not defined, it will apply no filter.

Let's imagine we want to run Karyon and we want our plots to use a window size of 2Kbp and up to 50 scaffolds represented in the plots that must be longer than 10Kbp. The command will be:

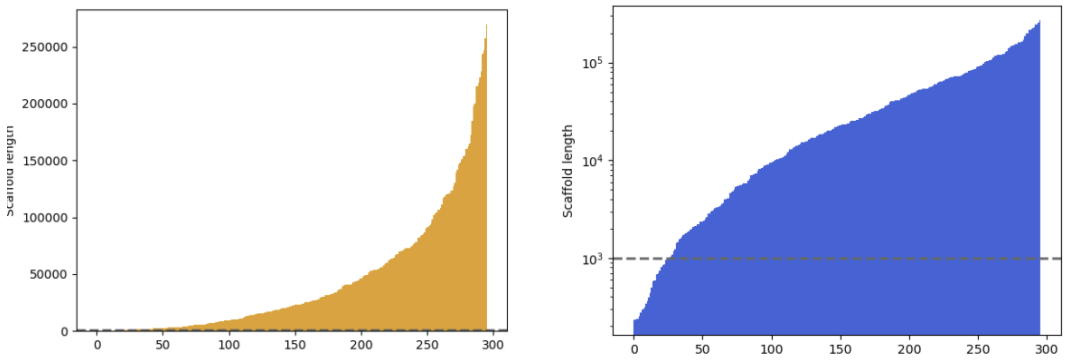
```
python3 karyon.py -l L1_1.fastq L1_2.fastq L2_1.fastq L2_2.fastq -d karyon_results -w 2000 -x 50 -s 10000
```

You can also generate the plots as a standalone script. For that, you have to call the script allplots.py, which requires the user to provide a fasta file, a bam file, a vcf file, a mpileup file and the fastq library used to create it. Of course, allplots.py has the same options for configuring the plots as karyon.py. Let's assume we are using the previous example, where we have run Karyon with the prefix "example" for the output files:

```
python3 allplots.py -f example.fasta -v example.raw.vcf -p example.mpileup -b example.sorted.bam -l L1_1.fastq
```

8. How to interpret the results

7.a.Scaffold length plots



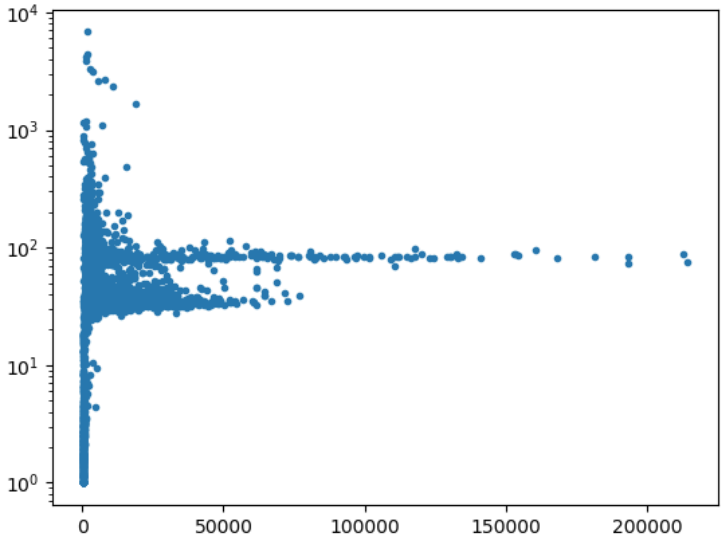
These plots represent the distribution of scaffold length. Karyon generates the results in linear and logarithmic distribution. Very short scaffolds (shorter than 1Kbp) might introduce noise in the analyses and might be interesting to just filter them out. Dashed line is equal to window size, and can be configured with the flag `-w` or `--window_size`. Karyon will generate two plots, one named [prefix].lenlin.png and [lenlog.png], which correspond to the length distribution in linear format (yellow) and logarithmic format (blue).

7.b. Scaffold versus coverage

Karyon will generate a scatter plot representing the average coverage versus length for each scaffold in the assembly. Quite often short scaffolds have different coverage from the majority of the genome, which might be indicative of contamination or repetitive regions. Even more, long scaffolds might present a range of coverage, which might be indicative of aneuploidy. This is generated in the file [prefix].len_v_cov.png.

In this example, assembly is highly fragmented, but it clearly shows two populations with different coverage.

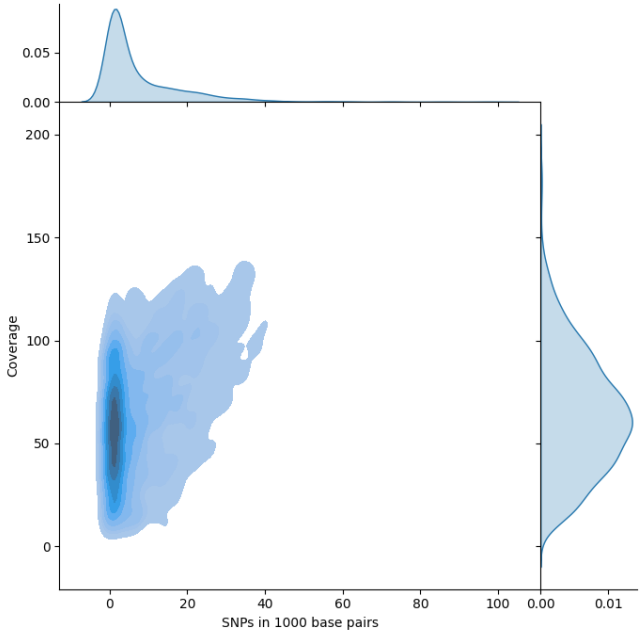
7.c. Variation vs. coverage



This plot allows the user to observe overall patterns across the whole genome. It uses a Kernel Density Estimation over a cloud of dots. Each dot represents the number of heterozygous or homozygous sites (X axis) versus the average coverage (Y axis) in a window of the genome, typically 1Kb. These dots will tend to form one or more populations. Here we compile some of the most common simple patterns:

1. **Cloud clustered towards 0 in the X axis:**
   This is the common pattern observed for a haploid or highly homozygous organism. The average Y value of the cloud corresponds to the average coverage of the genome.
2. **Cloud that diffuses across the X axis, but with a stable Y value.**
   This situation is typical of diploids and heterokaryons. Heterozygosity is rarely evenly distributed across the genome. The average Y value of the cloud corresponds to the average coverage of the genome. If part of the cloud clusters at zero heterozygosity and presents the same coverage, that represents the fraction of the genome that is homozygous, which might arise either from recombination or correspond to slowly evolving regions.

3. **Diffuse cloud across both X and Y axes.**



This type of distribution is often caused by the presence of contaminating sequences, especially if such contamination is heterogeneous in nature. If these patterns are observed, werecommend running BlobTools (https://github.com/blobtoolkit/blobtools2) on the sample.

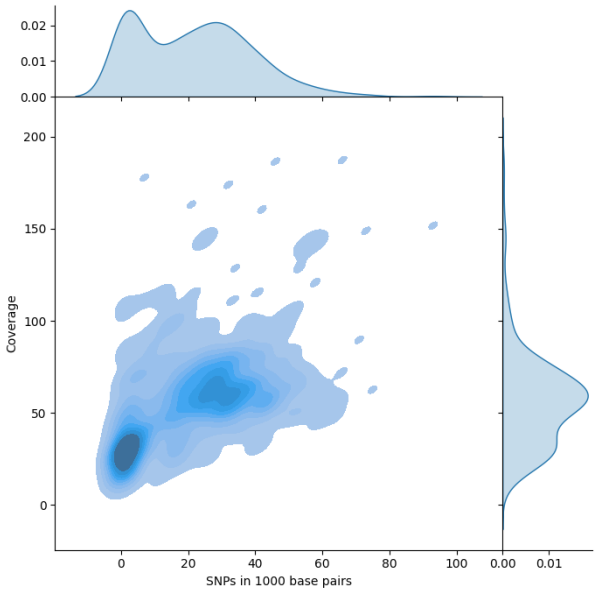
4. **Cloud clustered towards 0 in the Y axis:**
   This pattern might also indicate the presence of contaminating sequences in the genome assembly. Contamination might affect only some of the libraries used. If the graphics are generated using one of the non-contaminated libraries, regions of the assembly that were assembled from contaminating reads will not appear. If these patterns are observed, we recommend using BlobTools to identify the contaminating libraries, and then try assembling the genome without those problematic libraries.

It is important to note that quite often more than one cloud will appear, which creates composite distributions. Here we comment some of the most common ones:

1. **Homozygous cloud + Heterozygous cloud clustered across the same Y values:**
   This pattern indicates that some regions in the genome have low heterozygosity. Depending on the relative size of each cloud, these homozygous regions might correspond to slowly evolving sites. However, if the two clouds are well defined and separated, it typically indicates loss of heterozygosity. This means that the organism is a diploid formed by the combination of two divergent parentals,that has secondarily lost

regions corresponding to one or the other parental due to chromosome recombination. This pattern is common in species that formed due to hybridization events.

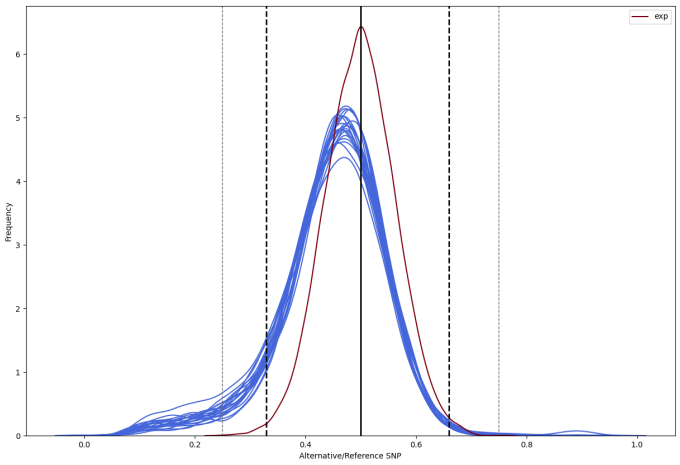
2. **Two or more clouds clustered at different Y values:**



In these situations the important factor to identify is the relative Y values of these two clouds. Often, the highest Y values will be close to double or triple the smallest, which is indicative of different ploidy levels. This situation is common in aneuploids, and we recommend checking the plots per scaffold to certify that some scaffolds are driving the observed behavior. Typically, clouds with higher Y values will tend to have higher and more diffuse X values, since higher ploidy levels contain more sites that can vary. If the relative proportions of the Y peaks are incompatible with aneuploid levels, it is likely that the dual behavior is caused by a single, highly abundant second source of DNA, such as a symbiont. Such scenarios can be identified easily with BlobTools. Alternatively, diffuse clouds with very high coverage might be indicative of a high fraction of repetitive elements.

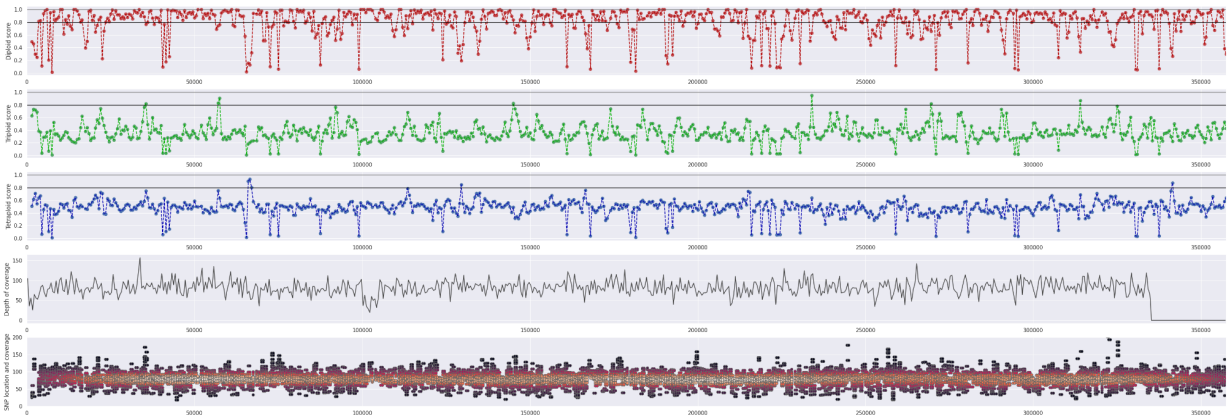
7.d. Fair coin plot

This plot represents the proportion of alternative vs. reference SNP for the whole genome and for each individual scaffold. Vertical lines indicate expected frequencies of 0.5, 0.33 and 0.25; corresponding to ideal diploids, triploids and tetraploids, respectively. An expected frequency is drawn, which is based on a per-site simulation of proportions assuming ideal 0.5 relative frequencies and random sampling equal to the coverage of the site.



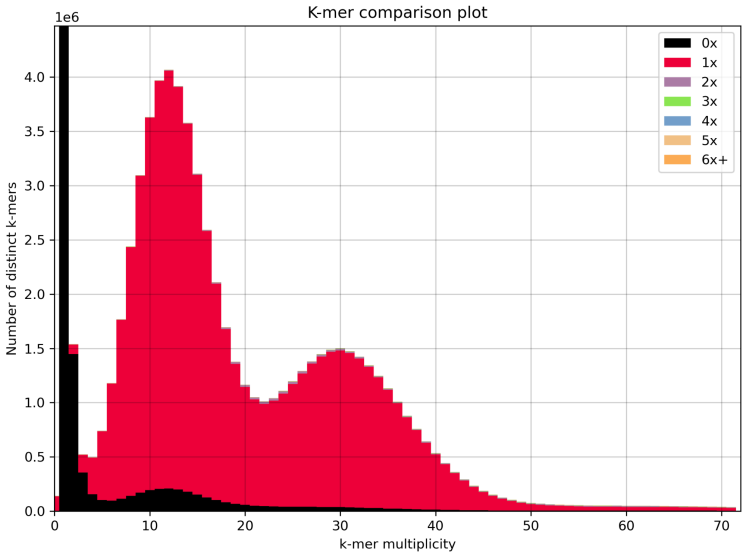
In a diploid or heterokaryotic organism, most of the scaffolds will present a frequency maximum close to 0.5 and the curve will approach the theoretical ideal. Repetitive elements, stochastic calling errors (variant calling algorithms have a bias against the alternative SNP) and other methodological limitations might slightly deviate from the ideal curve. Triploid regions will present two maxima around 0.33 and 0.66. While tetraploid regions could theoretically present three maxima, at 0.25, 0.5 and 0.75, in practice the relative signal will depend greatly on the composition of the subgenomes. For example, an allotetraploid formed by hybridization of two highly homozygous diploids might be indistinguishable from a regular diploid and present most signal at 0.5. Analogously, a second generation allotetraploid hybrid between an AB hybrid and an AA homozygous will present most of its heterozygosity in the 0.25 and 0.75 mark. It is important to note that the different proportion of reads for each subgenome might cause biases against the minority subgenomes during genome assembly. Finally, some heterokaryons might present systematic deviations from the expected distributions that might be visible in this plot as slight, mostly symmetric deviations from the theoretical distribution. Detection of heterokaryosis is far from trivial, and experimental approaches might be necessary to verify it.

7.e. nQuire per scaffold plot



Karyon will run nQuire (https://github.com/clwgg/nQuire) across sliding windows of defined length (by default 1Kbp) across different scaffolds. The plot for each scaffold contains five subplots. The first three represent the nQuire score for diploid, triploid and tetraploid for that particular window. This allows the user to visualize patterns of aneuploidy per scaffold, especially with regards to diploid and triploid regions. The fourth and fifth subplot represents the location and coverage of SNPs across the scaffold. A color code is assigned to represent the density. Since nQuire requires information of SNPs, homozygous regions cannot be assessed and will appear as missing data.

7.f. K-mer distribution



Karyon is able to call KAT (https://kat.readthedocs.io/en/latest/) to analyze the distribution of unique k-mers in the favourite library. The shape of the curve is informative about the ploidy level of the sample and the number of peaks typically corresponds with the maximum ploidy level. One of the main factors that affect the relative size of the peaks is loss of heterozygosity, as this reduces the amount of heterozygous positions. Additionally, aneuploidies might appear as peaks if they affect a large enough proportion of the genome. When interpreting these results it is important to check the proportion of mapping reads. A low proportion of mapping reads might be indicative of incomplete assembly, which might arise due to some artifacts. Karyon will automatically generate a report using samtools flagstats. Ideally, the percent of mapping reads should be above 90%.

Due to the reduction step, k-mer distribution plots will often have a significant portion in black. This corresponds to k-mers that appear 0 times in the reference genome, which would be the genomic fraction reduced by Redundans. We can use this to have a visual representation of how much reduction has occurred in the genome, but only if the percent of mapped reads is large enough (samtools flagstats report). A low percent of mapped reads might indicate incomplete assembly or presence of contaminants (i. e. mitochondrial DNA).