isomut2py Documentation

Release 2.0.1

Orsolya Pipek

Feb 23, 2019

Contents

IsoMut2py is an easy-to-use tool for the detection and postprocessing of mutations from raw NGS sequencing data. It takes sets of aligned short reads (BAM files) as its input and can explore and compare the karyotypes of different samples, detect single nucleotide variations (SNVs), insertions and deletions (indels) in single or multiple samples, optimize the identified mutations whenever provided with a list of control samples, plot mutation counts and spectra on readily interpretable charts and decompose them to predefined reference signatures.

IsoMut2py is an updated version of the original IsoMut software, mainly implemented in python. The most time-consuming parts of the workflow are however written in C, but also accessible through python wrapper functions.

Contents:

CHAPTER 1

Introduction

1.1 IsoMut2py: a python module for the comprehensive detection and analysis of genomic mutations

IsoMut2py is an easy-to-use tool for the detection and postprocessing of mutations from raw NGS sequencing data. It takes sets of aligned reads (BAM files) as its input and can explore and compare the karyotypes of different samples, detect single nucleotide variations (SNVs), insertions and deletions (indels) in single or multiple samples, optimize the identified mutations whenever provided with a list of control samples, plot mutation counts and spectra on readily interpretable charts and decompose them to predefined reference signatures.

IsoMut2py is an updated version of the original IsoMut software, mainly implemented in python. The most time-consuming parts of the workflow are however written in C.

1.2 Features

- easy installation with dependencies using `pip`
- **karyotype exploration** for a single sample, using a Bayesian approach
- **karyotype comparison** between sample pairs, for a naive identification of CNVs
- **karyotype plots**, **coverage histograms**
- **SNV** and **indel** detection in **single or multiple samples**
- detection of both **unique and shared mutations**
- refined mutation detection based on local ploidy information
- **automatic optimization** based on the user-defined list of control samples with easily interpretable figures as sanity checks
- option for loading and filtering a preexisting set of mutations
- **basic hierarchical clustering** of samples based on the number of shared mutations

- **plots of SBS, DBS and ID spectra** source
- **decomposition of SBS, DBS and ID spectra** to a mixture of reference signatures using expectation maximization
- **signature composition plots**
- straightforward querying of details of samples in mutated positions

1.3 Dependencies

1. **samtools**: In order to use the functions for mutation calling or ploidy estimation, samtools needs to be installed. However, plotting and filtering of mutations is available without samtools.
2. pandas
3. numpy
4. scipy
5. matplotlib
6. pymc3
7. theano
8. seaborn
9. biopython

Other than **samtools**, all dependencies can be automatically installed using `pip`.

1.4 Authors

Most of the code has been written by Orsolya Pipek, although the C code directly inherited from the original IsoMut software has been written by Dezso Ribli.

The whole project was done in collaboration of:

- Department of Physics of Complex Systems, Eotvos Lorand University (Orsolya Pipek, Dezso Ribli, Istvan Csabai)
- Institute of Enzymology, Research Centre for Natural Sciences, Hungarian Academy of Sciences (Adam Poti, David Szuts)
- Center for Biological Sequence Analysis, Department of Systems Biology, Technical University of Denmark (Zoltan Szallasi)

Implementation of the Fisher's exact test in C was borrowed from Christopher Chang.

1.5 How to cite

Coming soon.

CHAPTER 2

Getting started

2.1 Installation

IsoMut2py can be easily installed with `pip`, Python's own package manager:

```
pip install isomut2py
```

This also installs all python dependencies, *but not samtools*. To install `samtools` on your computer, follow the instructions here.

Note: If you only wish to perform postprocessing and plotting steps on a preexisting set of mutations, you can skip the installation of `samtools`.

2.2 Basic mutation detection on a set of 10 samples

2.2.1 Preparations, downloading an example dataset

As a first step we import IsoMut2py, which also compiles the C scripts used for mutation detection.

[1]:
```
import isomut2py as im2
```

```
Compiling C scripts...
Done.
```

Next, we download an example dataset of 10 samples from the DT40 cell line. (The BAM files in this dataset only contain reads that have been aligned to the first chromosome of the reference genome for faster mutation detection.)

The variable `exampleDataDir` stores the path to the directory where you wish to download the example dataset. Make sure to sets its value appropriately.

Note: Downloading can take a while, if you have the example data already downloaded, skip this step.

[2]:
```
exampleDataDir = 'isomut2py_download_dir/'
```

[3]:
```
im2.examples.download_example_data(path=exampleDataDir)
```

```
Downloading file from URL "http://genomics.hu/tools/isomut2py/
↪isomut2py_example_dataset.tar.gz" to isomut2py_download_dir/
↪isomut2py_example_dataset.tar.gz
File size: 2.1GiB
Downloading might take a while...
Download completed in 0 day(s), 11 hour(s), 38 min(s), 32 sec(s).
---------------------------
Extracting downloaded file to isomut2py_download_dir
Extracting completed in 0 day(s), 0 hour(s), 0 min(s), 59 sec(s).
```

And now we load the basic settings for the example dataset into a directory with:

Note: Make sure to set the value of the `exampleResultsDir` to a path where fairly large temporary files can be stored.

[4]:
```
exampleResultsDir='isomut2py_results_dir'
```

[5]:
```
param = im2.examples.load_example_mutdet_settings(example_data_path = exampleDataDir,
                                                  output_dir = exampleResultsDir)
```

2.2.2 Mutation detection

Mutation detection can be performed with the commands below.

[6]:
```
mutDet = im2.mutationcalling.MutationCaller(**param)
mutDet.run_isomut2_mutdet()
```

```
File no_ploidy_info does not exist. Using constant default ploidy of 2.
2018-12-05 10:30:57 - Mutation detection with IsoMut2


        2018-12-05 10:30:57 - Running IsoMut2 without local realignment:


                2018-12-05 10:30:57 - Preparations for parallelization:

                        2018-12-05 10:30:57 - Defining parallel blocks ...
                        2018-12-05 10:30:57 - Done

                2018-12-05 10:30:57 - (All output files will be written to␣
↪isomut2py_results_dir)

                2018-12-05 10:30:57 - Generating temporary files, total number of␣
↪blocks to run: 101

                2018-12-05 10:30:57 - Currently running: 1 2 3 4 5 6 7 8 9 10 11 12␣
↪13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40␣
↪41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68␣
↪69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96␣
↪97 98 99 100 101
```

(continues on next page)

(continued from previous page)

```
                2018-12-05 10:33:00 - Finished this round.

                2018-12-05 10:33:00 - Finalizing output...
                2018-12-05 10:33:00 - Cleaning up temporary files...

2018-12-05 10:33:01 - IsoMut2 finished. (0 day(s), 0 hour(s), 2 min(s), 4 sec(s).)
```

2.2.3 Further optimization

The results of the mutation detection are stored in files `all_SNVs.isomut2` and `all_indels.isomut2` in `exampleResultsDir`. These are essentially raw, unfiltered results of the mutation detection pipeline that can be refined with the `optimize_results()` command. For this, one has to supply a list of control samples where no unique mutations are expected to be found. (These can be starting clones of a sample group from which other samples are also available or normal samples of tumor-normal sample pairs or the same biological sample sequenced multiple times.)

In our case, file pairs (`S03.bam`, `S05.bam`) and (`S08.bam`, `S10.bam`) are two independent sequencing results of the same biological sample, thus no unique mutations should be present in any of these.

Setting these files as control samples and allowing no mutations in these samples filters other mutations as well, based on their score value:

[7]:
```
mutDet.optimize_results(control_samples=['S03.bam', 'S05.bam', 'S08.bam', 'S10.bam'],␣
↪FPs_per_genome=0)
```

2.2.4 Interpreting the results, figure generation

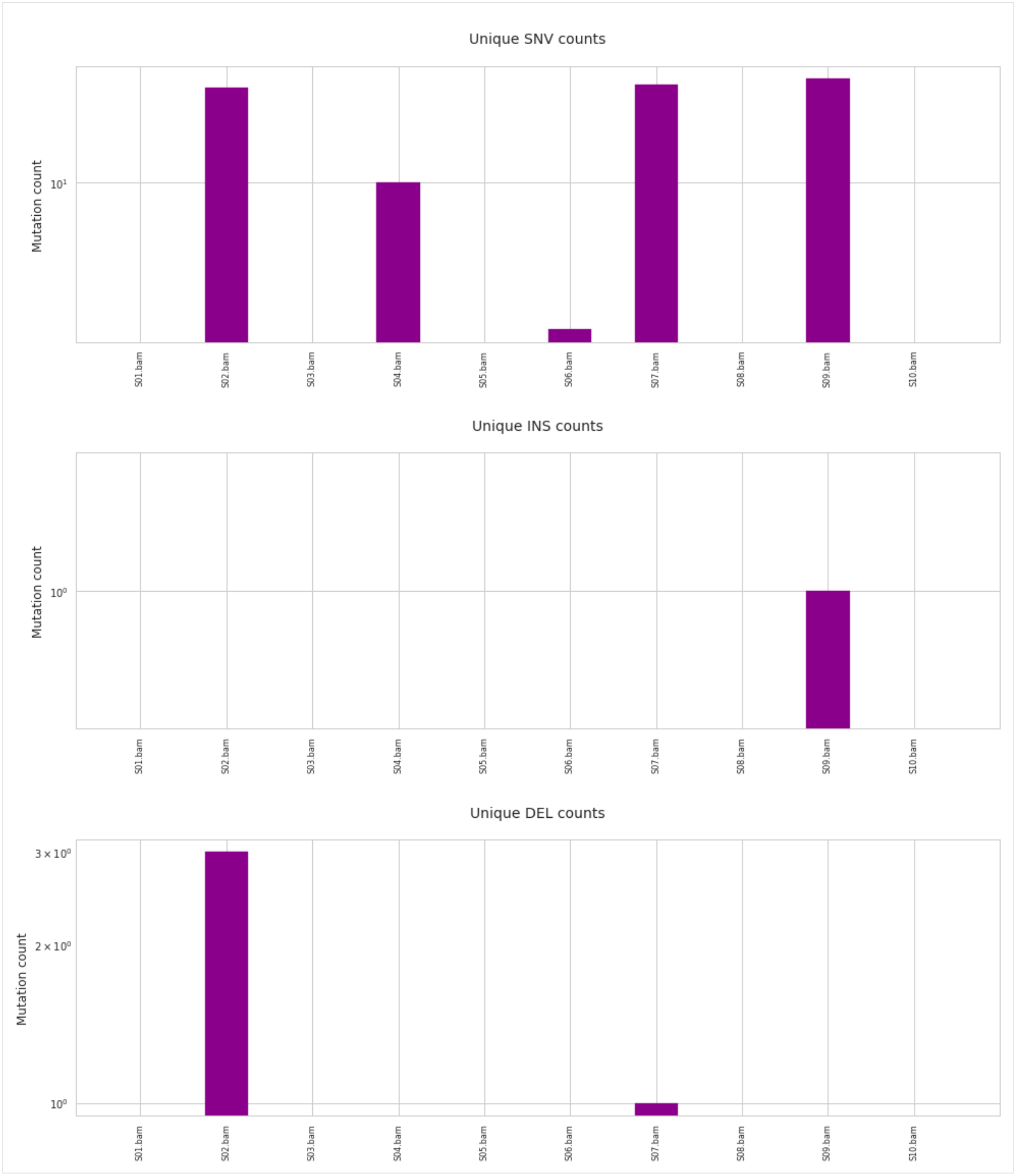
Now the final set of mutations can be further investigated.

First it is generally a good idea to check the number of (unique) mutations in each investigated sample. Plotting only unique mutations allows us to check that indeed none of these were found in control samples. This can be achieved with the following command:

Note: Calling the magic command `%matplotlib inline` simply ensures that figures are visible in the jupyter notebook environment.

[8]:
```
%matplotlib inline
```

[10]:
```
f = mutDet.plot_mutation_counts(unique_only=True)
```



Given that the investigation was confined to the first chromosome, very meaningful conclusions should not be drawn from this example. The number of indels is extremely low, but in the case of SNVs it is apparent that control samples lack unique mutations, while other samples do not.

Further plotting options and additional tools for interpreting the results are discussed in *Further analysis, visualization*.

An already existing set of mutation caller results can also be analysed with isomut2py. On how to work with external files, see *Importing external mutations*.

On how to perform ploidy estimation and use the results for fine-tuning mutation detection, see *Advanced ploidy estimation*.

CHAPTER 3

General use cases

3.1 General steps for analyzing various, differently treated cell lines

IsoMut2py works best if you have multiple samples that are isogenic, thus samples of the same essential genetic background that have been either treated with different chemicals or underwent different genomic modifications, for example to test different DNA repair pathways. However, it is possible to simultaneously analyze samples from multiple cell lines, just make sure that you have a few samples in each cell line group.

If any of your cell lines have aneuploid genomes, see *this point* to prepare them for mutation detection. Once each of your non-diploid cell lines have a genome-wise ploidy estimation, you can feed this information to IsoMut2py by creating a ploidy info file first as described *here*.

For mostly diploid cell lines, the above step can be skipped.

Now you can run the actual mutation detection in a similar manner as described in *this example*. For a complete list of available parameters, see the *documentation of the MutationCaller class*. IsoMut2py searches for mutations unique to each sample by default, but if you wish to uncover shared mutations as well, see *this point*. By default, the mutation detection is run only once without local realignment to decrease run time. This could introduce a low rate of false positives due to alignment errors. If you would like to filter these out, see *Using local realignment*.

Once the mutation detection is complete, a further optimization step is strongly encouraged to filter out false positive calls. This can be performed as described *here*. On the importance of control samples and optimizations, see *Optimization of mutation calls with control samples*.

If you happen to come across any suspicious mutation calls, you have the option to manually explore them in greater detail, as described in *Checking original sequencing data in ambiguous positions*.

Once you are satisfied with the final set of mutations, you can further analyze these to decipher mutational signatures, decompose mutational spectra to the weighted contribution of reference signatures, plot mutations on rainfall plots or perform a hierarchical clustering of the analyzed samples as described *here*.

3.2 Detecting mutations shared among samples

By default, IsoMut2py searches for mutations that are unique in a given sample, thus germline mutations are ignored. This is useful when one wishes to find mutations that arise randomly in samples as the result of a specific treatment, for example. However, in some cases it can be beneficial to also uncover mutations that are shared among different groups of samples. To do this, the `unique_mutations_only` parameter of the *MutationCaller object* should be set to `False`. This will result in an increased run time, but non-unique mutations will be detected as well.

Even if the `unique_mutations_only` parameter is set to `False`, those mutations that are present in all analyzed samples will not be printed by default. These only contain information about the shared differences of the analyzed samples compared to the reference genome, thus they are rarely meaningful. If it essential for your specific goal to uncover these mutations as well, set the `print_shared_by_all` parameter to `True`. This increases both memory usage and computation time.

3.3 Using local realignment

IsoMut2py first processes genomic positions by scanning through an mpileup file generated with the `samtools mpileup` command with the option `-B`, which prohibits the probabilistic realignment of the reads, thus maintaining the noise due to alignment errors. When positions are evaluated for possible mutations, the samples *not* containing the given mutation are expected to be extremely clean (defined by the parameter `min_other_ref_freq`), thus foregoing local realignment makes this criterion even stricter by making sure that the given position is not affected by alignment errors either.

However, it is also possible that the alignment noise introduced by skipping local realignment could appear as a real mutation in a given sample, while the noise level remains below the threshold in other samples, thus introducing false positive mutations. To get rid of these, IsoMut2py can be run a second time, now *with* local realignment, only on those positions that have been identified as potential mutations in the first run. Only those of them are kept that pass the necessarry filtering steps without the alignment noise as well. This second run of analysis can be initiated by setting the parameter `use_local_realignment` of the *MutationCaller object* to `True`. Note that this will greatly increase run time.

3.4 Optimization of mutation calls with control samples

IsoMut2py uses a set of hard filters (adjusted for local ploidy) to detect mutations. However, the results of the mutation calling pipeline can be greatly refined by performing an additional optimization step. The sequencing of control samples and using this optimization step is strongly encouraged, as this way instrument specific errors and offsets can be eliminated.

A control sample is essentially defined as a sample where no unique mutations are expected to be found. Sequencing the same DNA twice produces two control samples, as the mutations found in these should be present in their pair as well. Starting clone samples in an experiment also work as control samples, as all subsequent clones should have all their initial mutations, regardless of their treatment. When sequencing tumor-normal pairs, the normal samples can be used as controls, as germline mutations are expected to be shared with respective the tumor sample.

The list of control samples, along with the desired genome-wise false positive rate can be supplied to the *Mutation-Caller object* with the parameters `control_samples` and `FPs_per_genome`. (False positive rate is defined here as the number of false positives per the analyzed genome, so take the length of the genome into consideration when setting its value.)

The optimization step uses the score values assigned to the mutations by the initial mutation detection algorithm. (These scores are based on the p-values of a Fisher's exact test for the bases found in the "least mutated" mutated sample and the "least clean" clean sample. The higher the score value, the more likely it is that the given mutation is real.) During the optimization, the score value that maximizes the number of unique mutations found in non-control samples while keeping the number of unique mutations in control samples below the threshold defined by `FPs_per_genome` is sought. Once the optimal score value is found, the initial set of mutations is filtered with this threshold. (Whenever the ploidy is non-constant in the genome, the optimization is performed for regions of different ploidies separately, while assigning a number of acceptable false positives to each region based on their length, while keeping their sum below the user defined threshold.)

Whenever possible, the above described optimization step is strongly encouraged.

3.5 Checking original sequencing data in ambiguous positions

Even after *optimization*, it is entirely possible that you find some suspicious mutations in the final call set. For example, if you have samples from two different cell lines, a mutation in all of the samples from one cell line except the starting clone, and in none of the other samples would be unexpected. To manually check the sequencing information in these genomic positions to make an educated decision on keeping or discarding the mutation, you can use the `check_pileup` method of the *MutationCaller object*. This will return a pandas DataFrame containing condensed information of the joint mpileup file of all analyzed samples in the specified genomic positions. The resulting DataFrame can be conveniently filtered based on the values in its columns.

3.6 Analysing aneuploid cell lines

If any of your cell lines have aneuploid genomes, it is good practice to run an initial ploidy estimation on the starting clones of these cell lines. This can be done by following the steps described *here*. Given that different treatments or specific genomic alterations tend to keep the original structure of the genome intact, it is usually enough to perform this step only on one sample for each cell line. However, if you expect your samples to have vastly different genomic structures, even if originating from the same cell line, make sure to run the above described ploidy estimation for each of your samples separately.

CHAPTER 4

Importing external mutations

For instructions on installation and basic exploration steps, see *Getting started*.

It is possible to post-process lists of mutations with `isomut2py` that were otherwise generated with an external variant caller tool. Here we demonstrate how external VCF files can be loaded and further analysed.

As `isomut2py` expects `pandas` DataFrames as inputs to handle mutation lists, we will be using the `pandas` package for the loading of VCF files.

[2]:
```
import isomut2py as im2
import pandas as pd
%matplotlib inline
```

```
Compiling C scripts...
Done.
```

4.1 Importing VCF files using pyvcf

VCF files can be easily processed in python with the `pyvcf` package. This can be installed with:

```
pip install pyvcf
```

[7]:
```
import vcf
```

To convert VCF files to `pandas` DataFrames with columns that can be parsed by `isomut2py`, we first need to make sure to find the values of `sample_name`, `chr`, `pos`, `type`, `score`, `ref`, `mut`, `cov`, `mut_freq`, `cleanliness` and `ploidy`. In order to perform downstream analysis of mutations, fields `cov`, `mut_freq` and `cleanliness` can be left empty, but nonetheless have to be defined in the dataframe.

Here we are importing VCF files generated by the tool MuTect2 (of GATK). If another tool was used for variant calling, make sure to modify parser function below. The example VCF files are located at `[exampleDataDir]/isomut2py_example_dataset/ExternalMutations/mutect2`. (For instructions on how to download the example datafiles, see *Getting started*.)

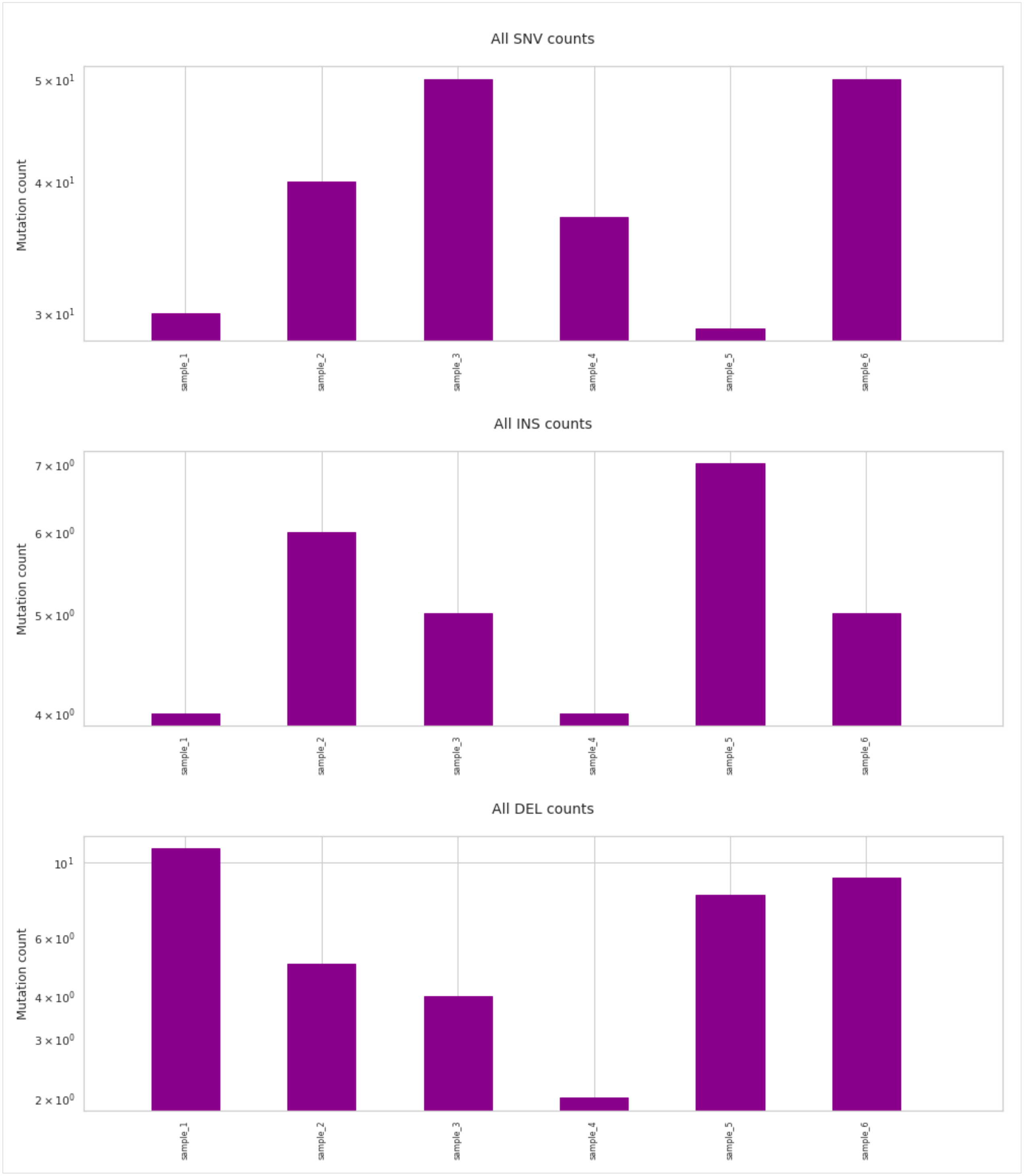
[91]:
```
def parse_VCF_record(record):
    if record.is_deletion:
        muttype = 'DEL'
    elif record.is_indel:
        muttype = 'INS'
    elif record.is_snp:
        muttype = 'SNV'

    return (record.CHROM, record.POS, muttype,
            record.INFO['TLOD'][0],
            record.REF, record.ALT[0],
            int(record.samples[0].data.DP),
            int(record.samples[0].data.AD[1])/int(record.samples[0].data.DP),
            int(record.samples[1].data.AD[1])/int(record.samples[1].data.DP))
```

[108]:
```
sample_dataframes = []
for i in range(1,7):
    vcf_reader = vcf.Reader(filename = exampleDataDir + 'isomut2py_example_dataset/
↪ExternalMutations/mutect2/'+str(i)+'_somatic_m.vcf.gz')
    d = []
    for record in vcf_reader:
        d.append(parse_VCF_record(record))
    df = pd.DataFrame(d, columns=['chr', 'pos', 'type', 'score', 'ref', 'mut', 'cov',
↪'mut_freq', 'cleanliness'])
    df['sample_name'] = 'sample_'+str(i)
    df['ploidy'] = 2
    sample_dataframes.append(df[['sample_name','chr', 'pos', 'type', 'score', 'ref',
                                'mut', 'cov', 'mut_freq', 'cleanliness', 'ploidy']])

mutations_dataframe = pd.concat(sample_dataframes)
```

Now we have all the mutations found in 6 samples in a single dataframe. This can be further analysed with the functions described in *Further analysis, visualization*. For example the number of mutations found in each sample can be plotted with:

[112]:
```
f = im2.plot.plot_mutation_counts(mutations_dataframe=mutations_dataframe)
```

```
Warning: list of control samples not defined.
```



4.2 Importing lists of mutations in arbitrary files

If your lists of mutations are stored in some kind of text files as tables, the easiest way to import them is to use the `pandas` python package. (As `isomut2py` heavily relies on `pandas`, it should already be installed on your computer.)

Here we merely include an example for such an import, make sure to modify the code and customize it to your table. (This data was generated with MuTect, that only detects SNPs, thus we don't expect to see any indels.)

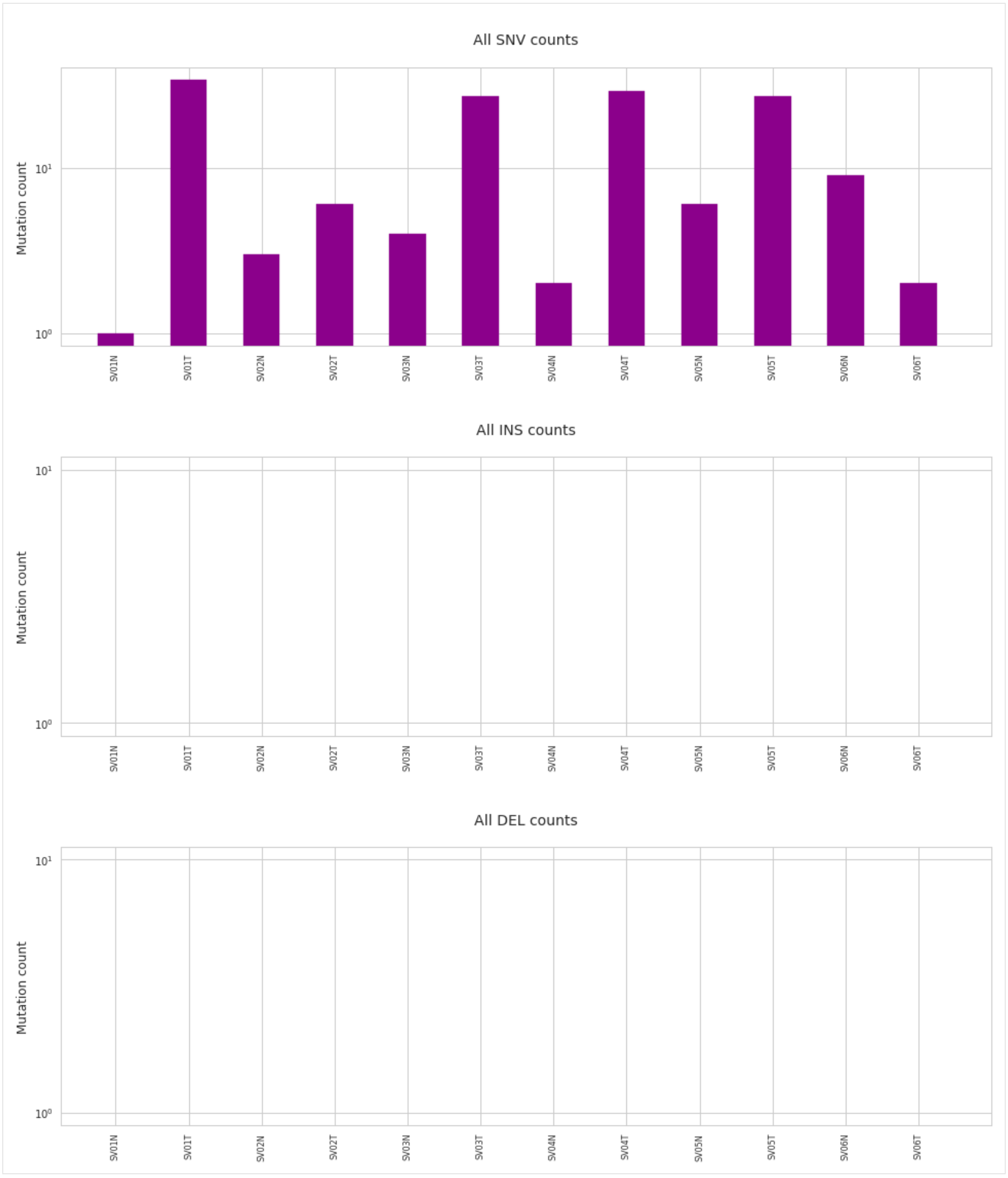
[4]:
```
exampleDataDir = '/nagyvinyok/adat83/sotejedlik/orsi/'
```

[27]:
```
sample_dataframes = []
for i in range(1,7):
    for sample_type in ['N', 'T']:
        df_sample = pd.read_csv(exampleDataDir + 'isomut2py_example_dataset/
↪ExternalMutations/mutect/SV0'+
                                str(i)+sample_type+'_chr1.vcf', sep='\t', comment='#')

        df_sample = df_sample[df_sample['judgement'] != 'REJECT']
        df = pd.DataFrame()
        df['chr'] = df_sample['contig']
        df['pos'] = df_sample['position']
        df['type'] = 'SNV'
        df['score'] = df_sample['t_lod_fstar']
        df['ref'] = df_sample['ref_allele']
        df['mut'] = df_sample['alt_allele']
        df['cov'] = df_sample['t_ref_count'] + df_sample['t_alt_count']
        df['mut_freq'] = df_sample['t_alt_count']/df['cov']
        df['cleanliness'] = df_sample['n_ref_count']/(df_sample['n_ref_count'] + df_
↪sample['n_alt_count'])
        df['ploidy'] = 2
        df['sample_name'] = 'SV0' + str(i) + sample_type
        df = df[['sample_name',
         'chr',
         'pos',
         'type',
         'score',
         'ref',
         'mut',
         'cov',
         'mut_freq',
         'cleanliness',
         'ploidy']]
        sample_dataframes.append(df)

mutations_dataframe = pd.concat(sample_dataframes)
```

[30]:
```
import warnings
warnings.filterwarnings("ignore")
```

[31]:
```
f = im2.plot.plot_mutation_counts(mutations_dataframe=mutations_dataframe)
```

```
Warning: list of control samples not defined.
```


All SNV counts
Mutation count
10¹
10⁰
SV01N
SV01T
SV02N
SV02T
SV03N
SV03T
SV04N
SV04T
SV05N
SV05T
SV06N
SV06T
All INS counts
All DEL counts

CHAPTER 5

Advanced ploidy estimation

For instructions on installation and basic exploration steps, see *Getting started*.

```
[1]: import isomut2py as im2
     %matplotlib inline
```

```
Compiling C scripts...
Done.
```

5.1 Ploidy estimation from scratch

For a complete tutorial on how to perform ploidy estimation from a single bam file, we will be using the raw example dataset, that can be downloaded with:

```
[3]: im2.examples.download_raw_example_data(path=exampleRawDataDir)
```

```
Downloading file from URL "http://genomics.hu/tools/isomut2py/
↪isomut2py_raw_example_dataset.tar.gz" to isomut2py_download_dir/
↪isomut2py_raw_example_dataset.tar.gz
File size: 528.5MiB
Downloading might take a while...
Download completed in 0 day(s), 2 hour(s), 49 min(s), 55 sec(s).
---------------------------
Extracting downloaded file to isomut2py_download_dir
Extracting completed in 0 day(s), 0 hour(s), 0 min(s), 7 sec(s).
```

And ploidy estimation settings can be loaded with:

Note: Make sure to set the value of the `examplePEResultsDir` to a path where fairly large temporary files can be stored.

```
[5]: PEparam = im2.examples.load_example_ploidyest_settings(example_data_
     ↪path=exampleRawDataDir,
                                                              output_dir=examplePEResultsDir)
```

We can generate a PloidyEstimator object and run the ploidy estimation with:

```
[6]: PEst = im2.ploidyestimation.PloidyEstimator(**PEparam)
     PEst.run_ploidy_estimation()
```

```
2018-11-15 12:47:24 - Ploidy estimation for file A.bam


        2018-11-15 12:47:24 - Preparing for parallelization...
                2018-11-15 12:47:24 - Defining parallel blocks ...
                2018-11-15 12:47:24 - Done

        2018-11-15 12:47:24 - (All output files will be written to ../Documents/
↪isomut2py_example_data/PE_example_results_20181114)

        2018-11-15 12:47:24 - Generating temporary files, number of blocks to run: 107
        2018-11-15 12:47:24 - Currently running:  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15␣
↪16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43␣
↪44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71␣
↪72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99␣
↪100 101 102 103 104 105 106 107

        2018-11-15 12:47:55 - Temporary files created, merging, cleaning up...

        2018-11-15 12:47:55 - Estimating haploid coverage by fitting an infinite␣
↪mixture model to the coverage distribution...

                2018-11-15 12:59:18 - Raw estimate for the haploid coverage: 27.
↪046445720783716

        2018-11-15 12:59:18 - Fitting equidistant Gaussians to the coverage␣
↪distribution using the raw haploid coverage as prior...

                2018-11-15 13:06:27 - Parameters of the distribution are saved to: ../
↪Documents/isomut2py_example_data/PE_example_results_20181114/GaussDistParams.pkl

                2018-11-15 13:06:27 - Final estimate for the haploid coverage: 32.
↪45404396554027

        2018-11-15 13:06:27 - Estimating local ploidy using the previously determined␣
↪Gaussians as priors on chromosomes:  I II III IV V VI VII VIII IX X XI XII XIII XIV␣
↪XV XVI

        2018-11-15 13:10:14 - Generating final bed file...


        2018-11-15 13:10:15 - Generating HTML report...


2018-11-15 13:10:49 - Ploidy estimation finished. (0 day(s), 0 hour(s), 23 min(s), 24␣
↪sec(s).)
```

The ploidy estimation pipeline consists of the following main steps:

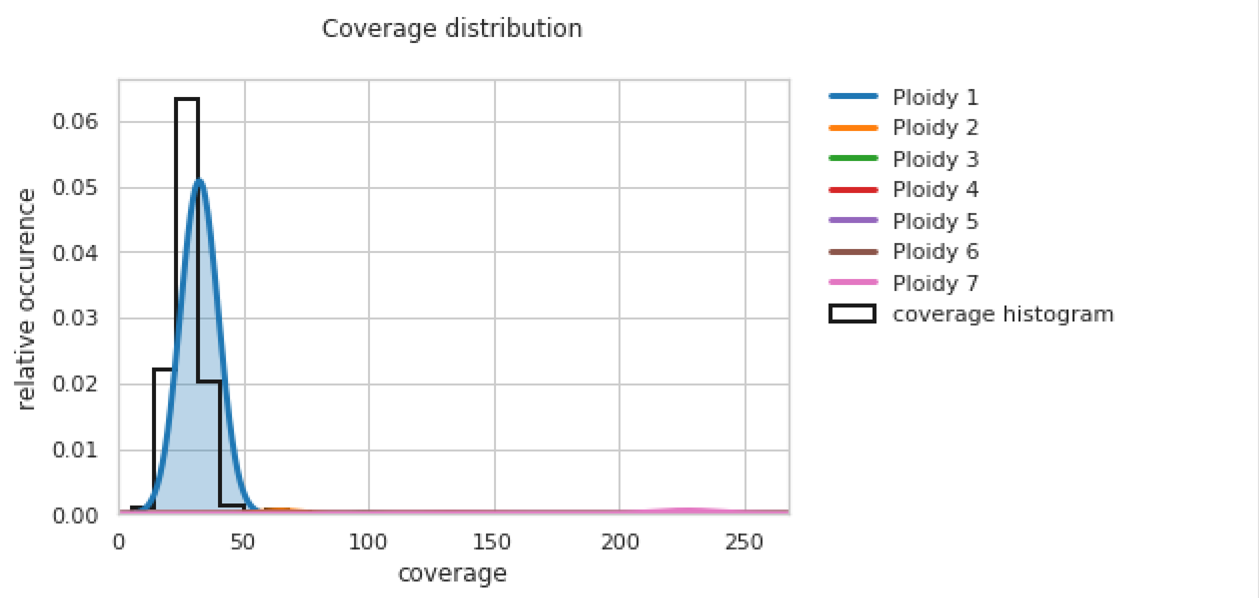
1. A temporary pileup file is generated from the original bam file which is scanned with a moving average method

to determine the reference allele frequency and the local average coverage for a set of investigated positions. This is performed in a highly parallelized manner, on short regions of the genome. Information collected for the genomic positions are temporarily saved to a set of intermediate files (one for each chromosome).

2. The average coverage of the haploid regions is estimated. This is achieved by fitting a many-component (20) Gaussian mixture model 10 times to the coverage distribution determined from the intermediate files. Using specific statistics of the parameters of the fitted distributions, an estimate for the haploid coverage is obtained. This whole step can be completely skipped by supplying an estimate value in the `user_defined_hapcov` attribute of the PloidyEstimator object. (If one has a reasonable idea about the haploid coverage (for example by checking sequencing results with a genome viewer), it is generally a good idea to set the `user_defined_hapcov` to this value to save time. For a strictly non-haploid genome, the haploid coverage is defined as the half of the diploid, the third of the triploid, etc. coverages.)
3. A seven component Gaussian mixture model is fitted to the coverage distribution with equidistant centers. The estimated (or user-defined) haploid coverage and its products with whole numbers are used as prior for the centers of the components, allowing the identification of seven distinct ploidies.
4. The seven component fitted distribution is used as prior to determine the ploidy of each genomic position in the intermediate files.

Thus the actual ploidy estimation largely depends on how accurately the seven component model fits the raw coverage distribution. It is good practice to check this by plotting them both on the same figure with:

```
[7]: f = PEst.plot_coverage_distribution()
```

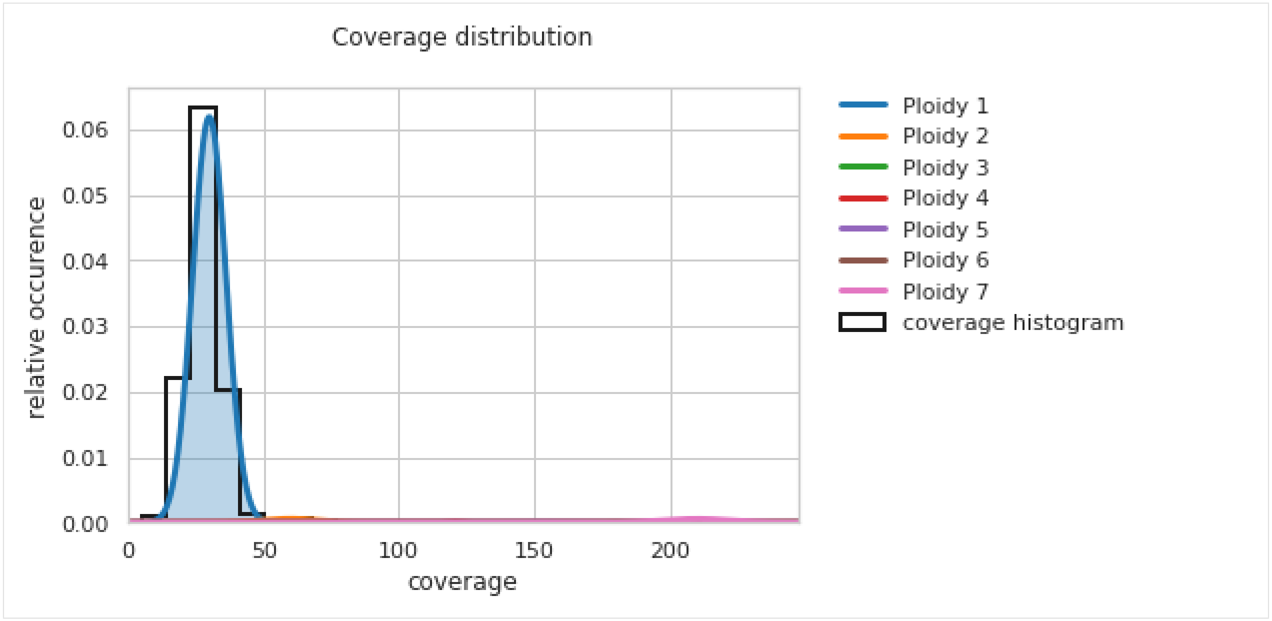


Given that the currently investigated sample is an almost fully haploid genome, having a single peak on the figure for the "Ploidy 1" curve is promising. However, the center of this curve is slightly shifted to the right.

If you are unhappy with the fit, estimate the haploid coverage manually, and try refitting the seven component model with:

```
[8]: PEst.fit_gaussians(estimated_hapcov = 25)
```

```
[9]: f = PEst.plot_coverage_distribution()
```



Naturally, you will need to rerun the final step of the ploidy estimation with the improved fit:

```
[10]: PEst.PE_on_whole_genome()
```

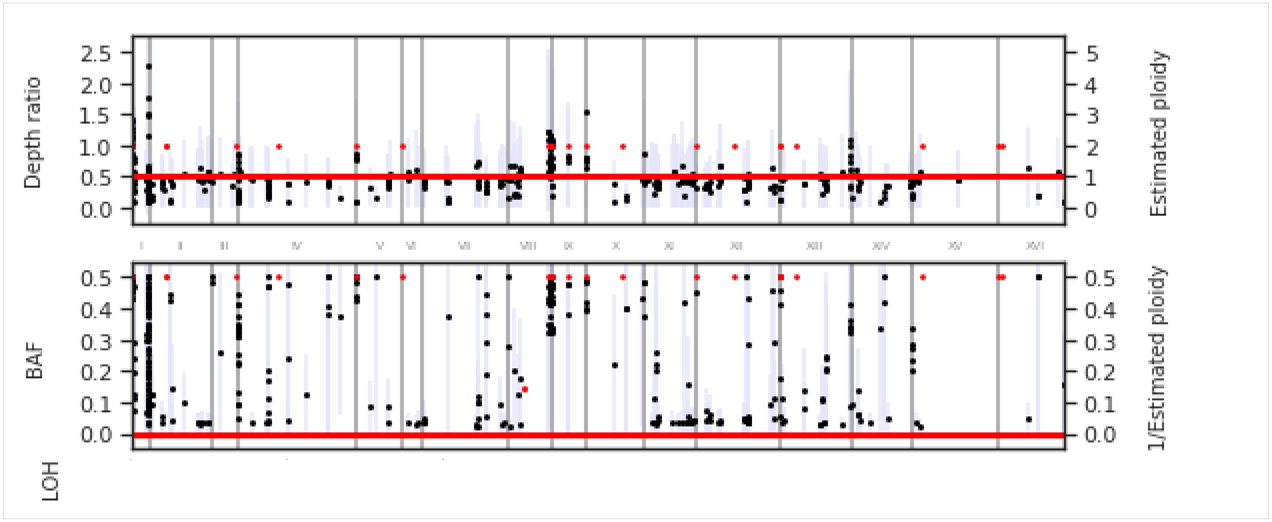
```
I II III IV V VI VII VIII IX X XI XII XIII XIV XV XVI
```

5.2 Visualizing karyotypes

To plot the results of ploidy estimation, isomut2py provides two function. A fairly concise summary of the results can be plotted with:

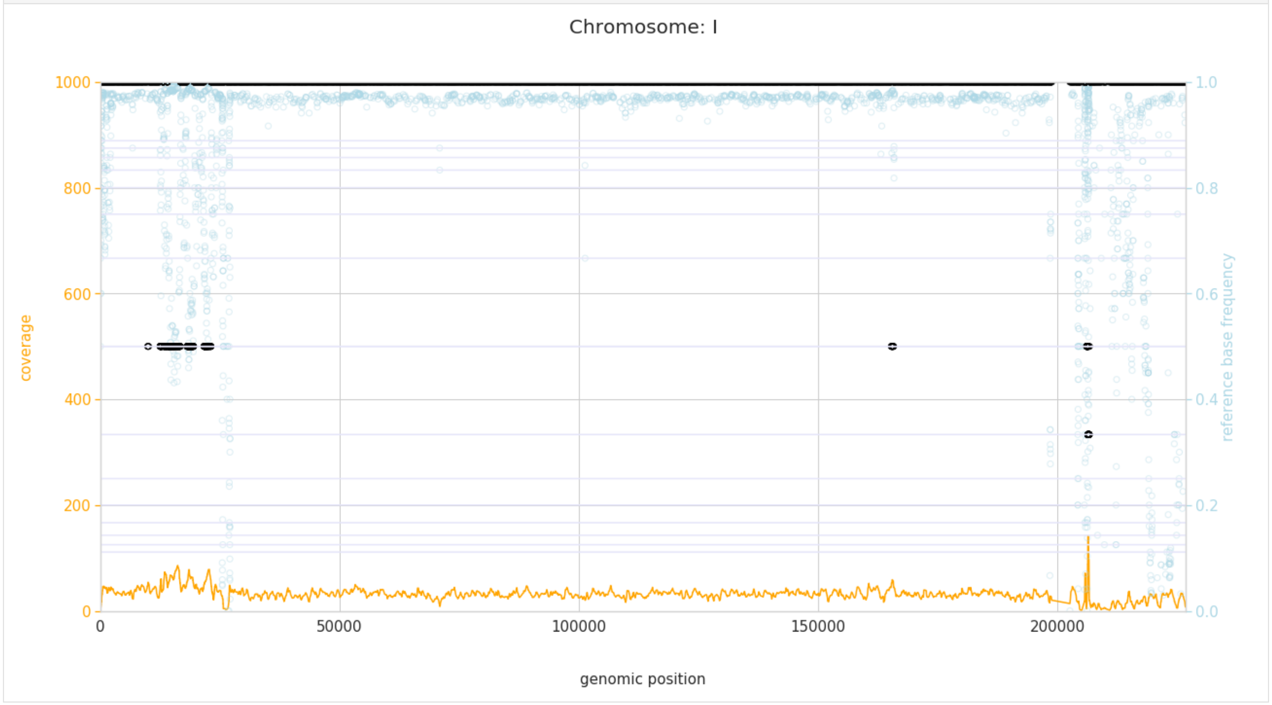
Note: The default values of `binsize`, `overlap` and `min_PL_length` are set to accomodate the plotting of large (human) genomes, thus for our current case, they should be adjusted to allow for a more easily interpretable figure. The value of `overlap` must always be smaller than `binsize`. These are used as shift- and windowsizes (respectively) for a moving average method that condenses the data to plottable points. The smaller their values are, the more points will appear on the figure, but the longer it will take to plot. Similary, `min_PL_length` controls the minimal length of a region with an unique ploidy to be plotted on the graph.

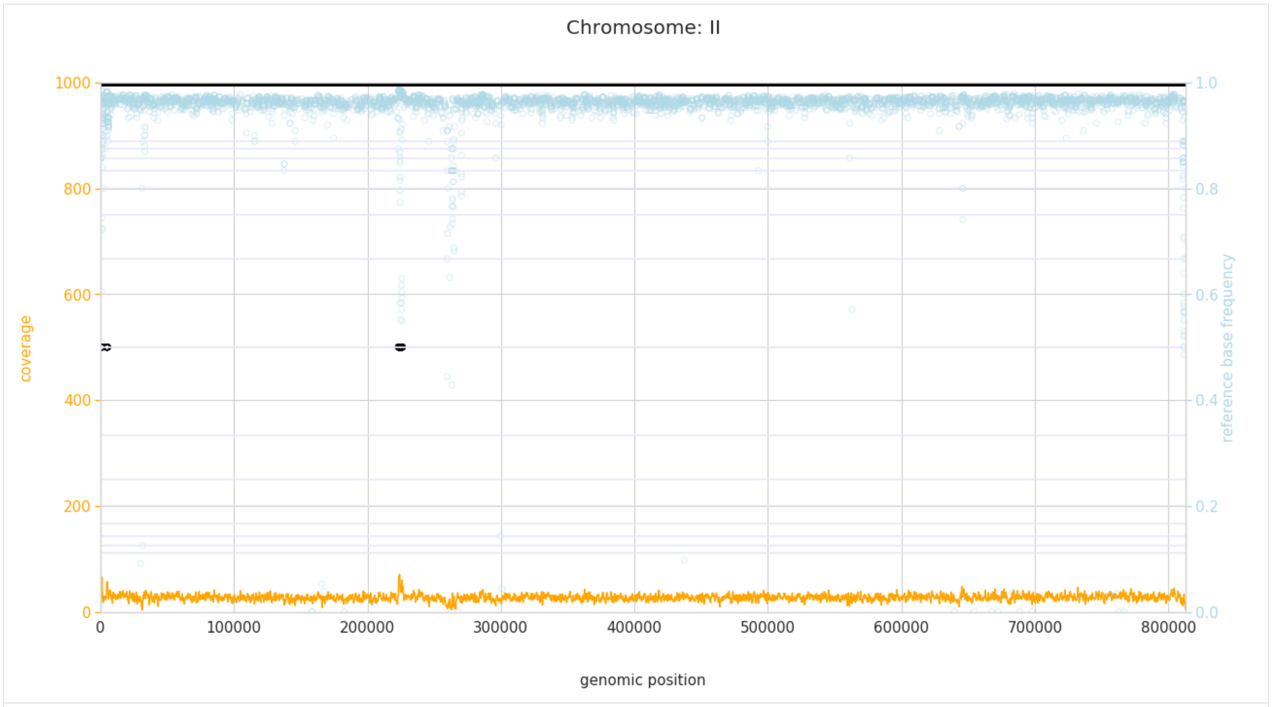
```
[11]: f = PEst.plot_karyotype_summary(binsize=100, overlap=10, min_PL_length=300)
```

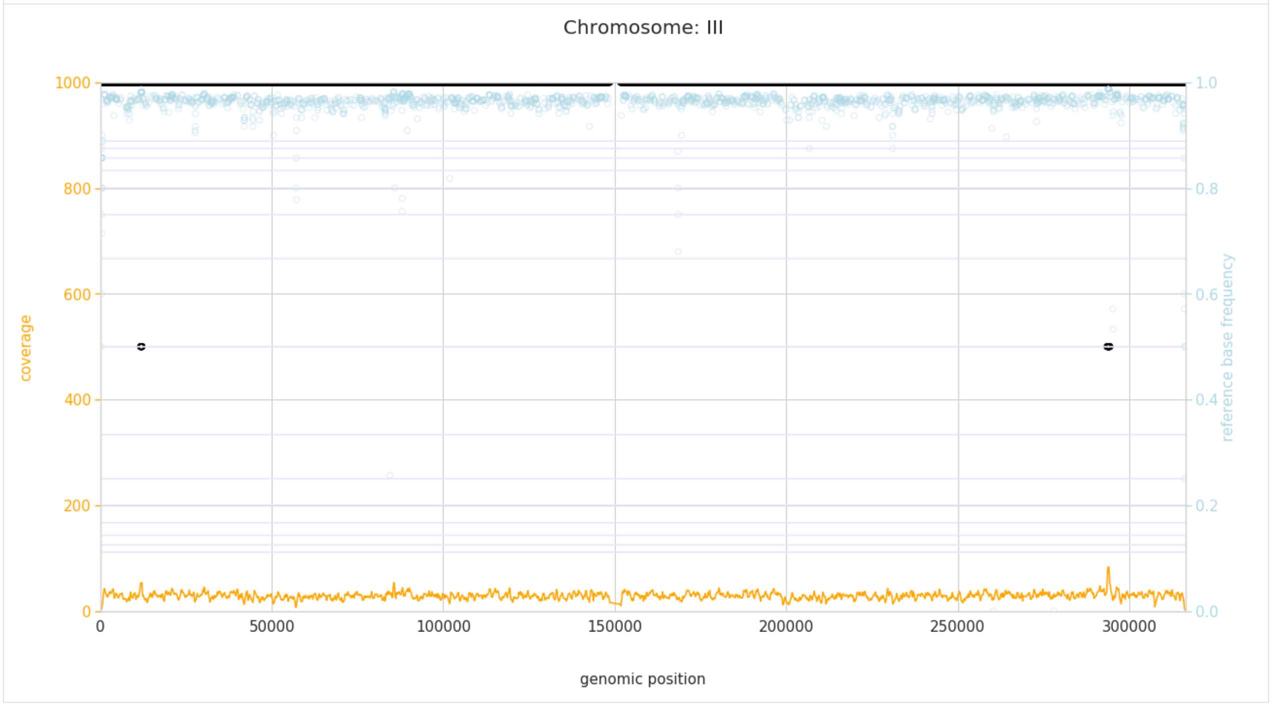


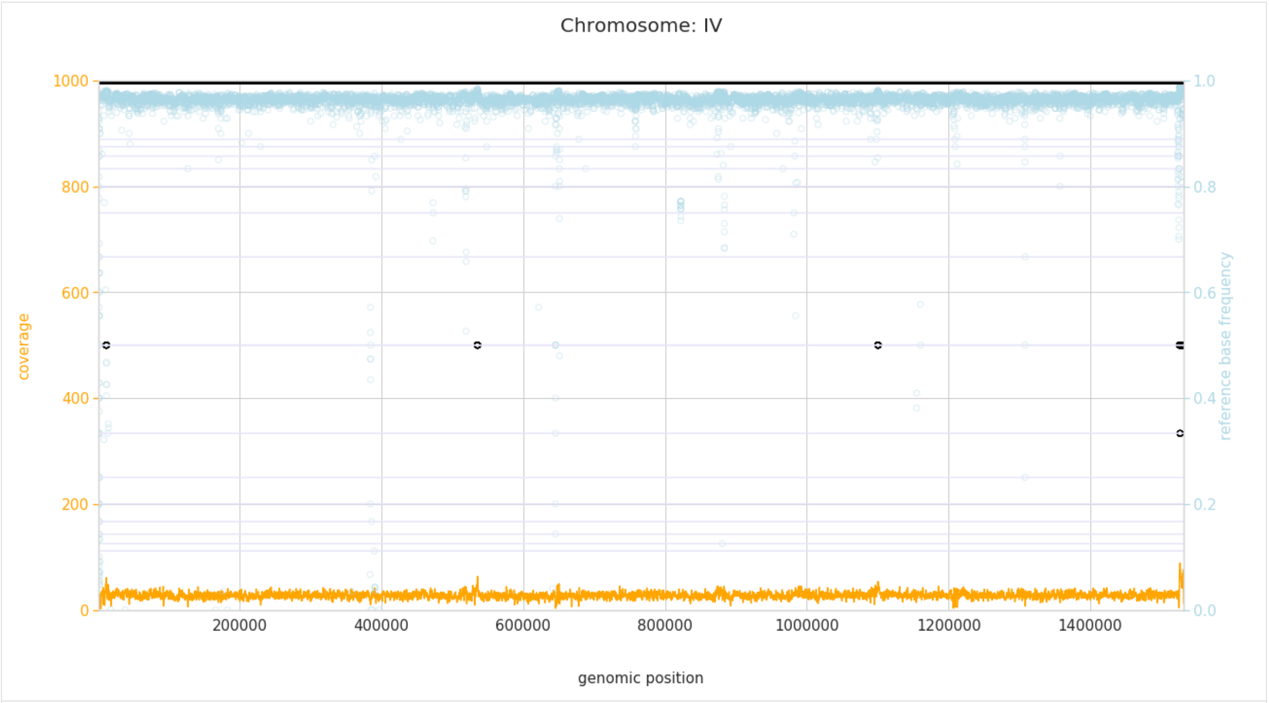
A more detailed plot of individual chromosomes can be generated with:

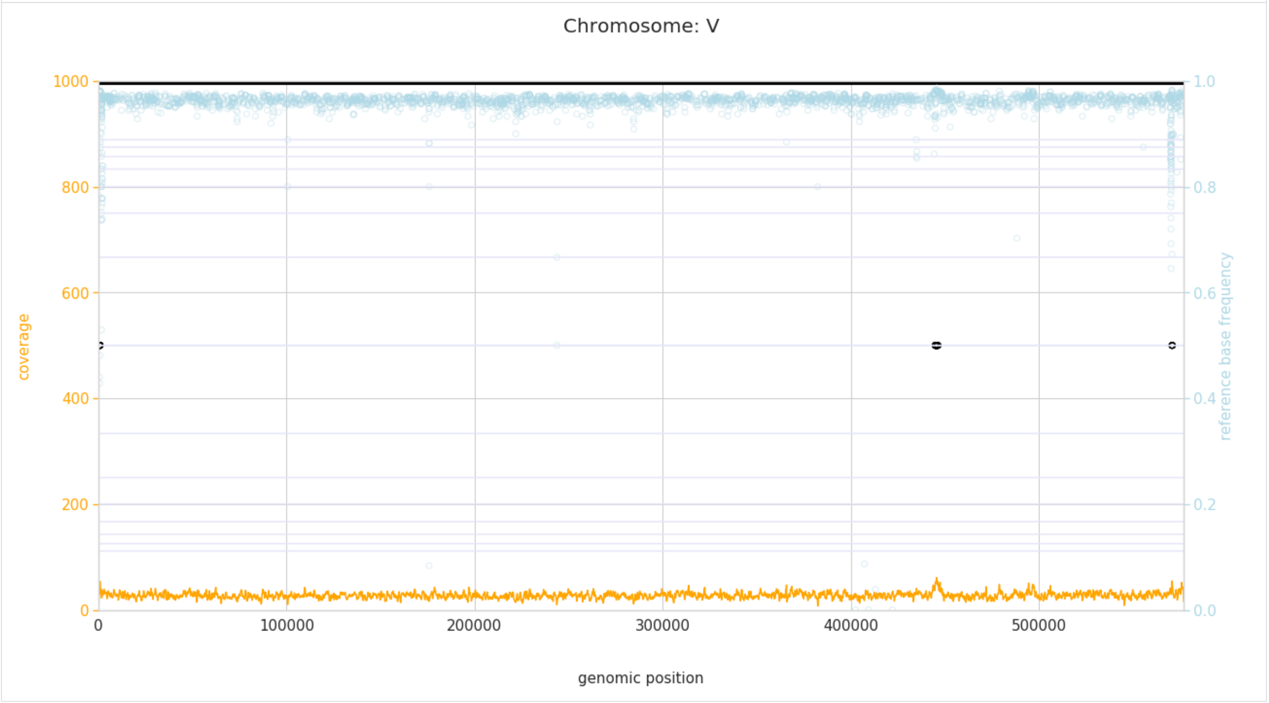
```
[12]: f = PEst.plot_karyotype_for_all_chroms()
```

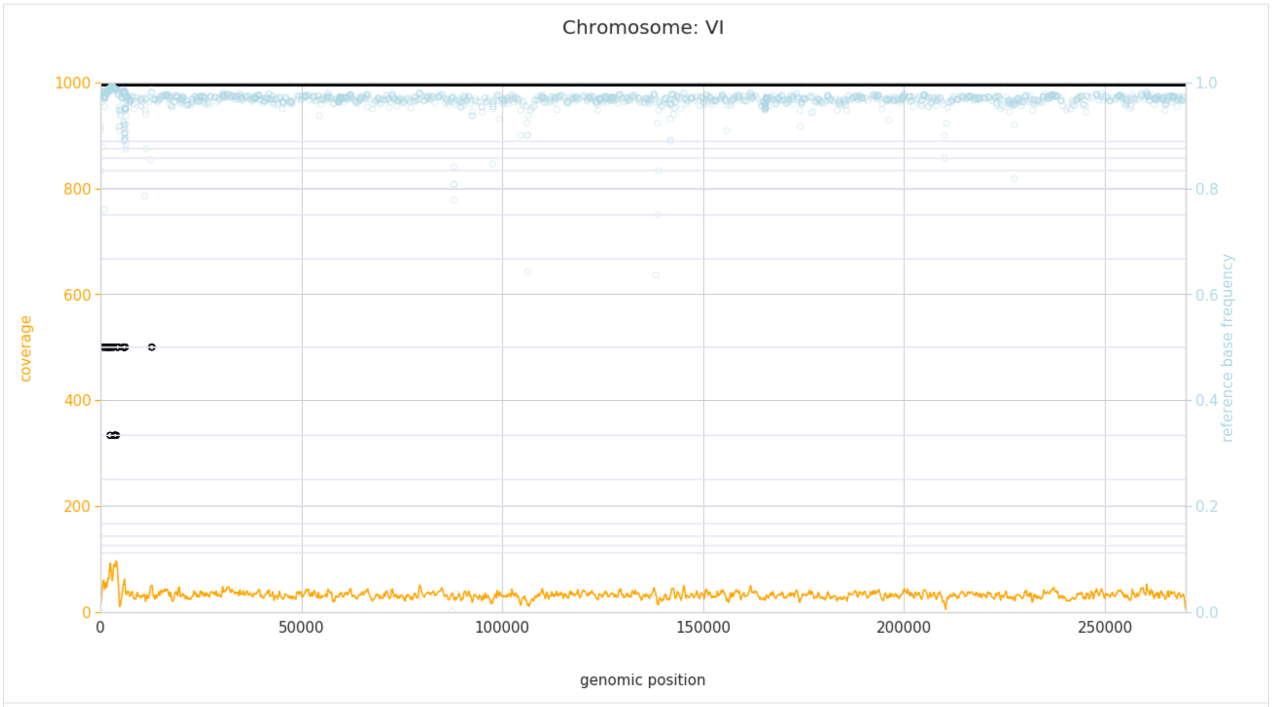


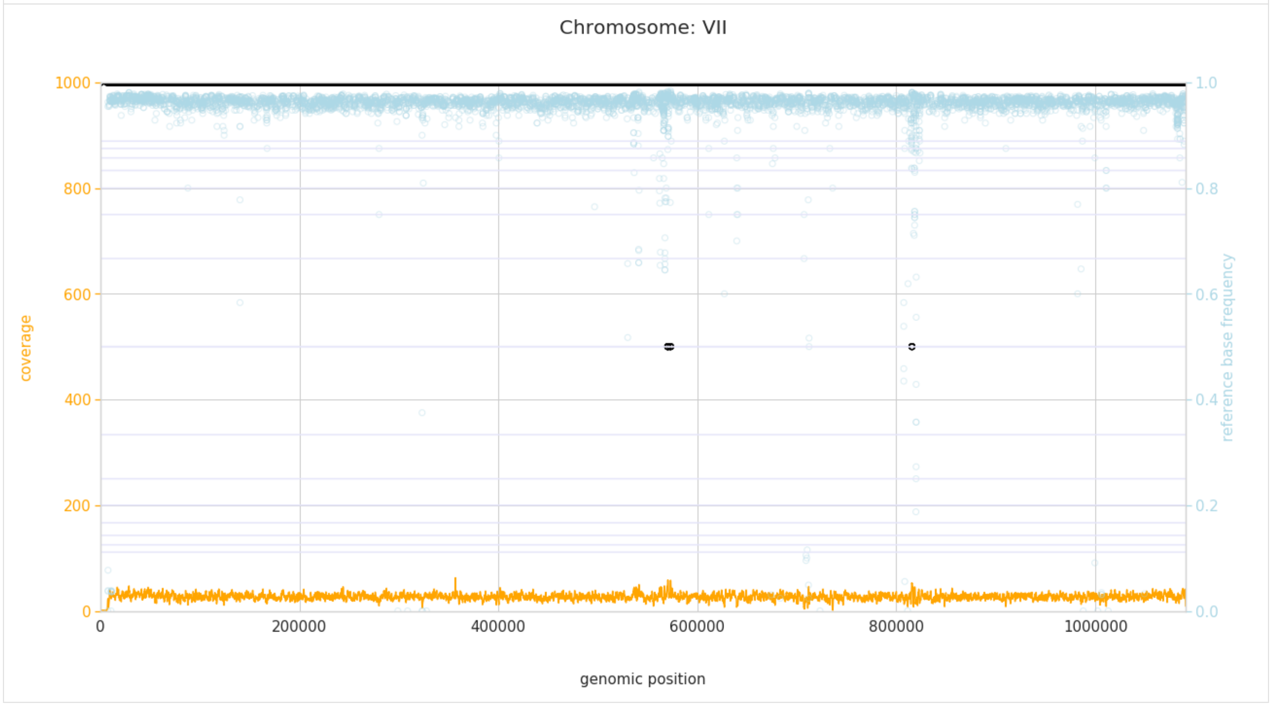

Chromosome: II
coverage
reference base frequency
genomic position
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
0
100000
200000
300000
400000
500000
600000
700000
800000

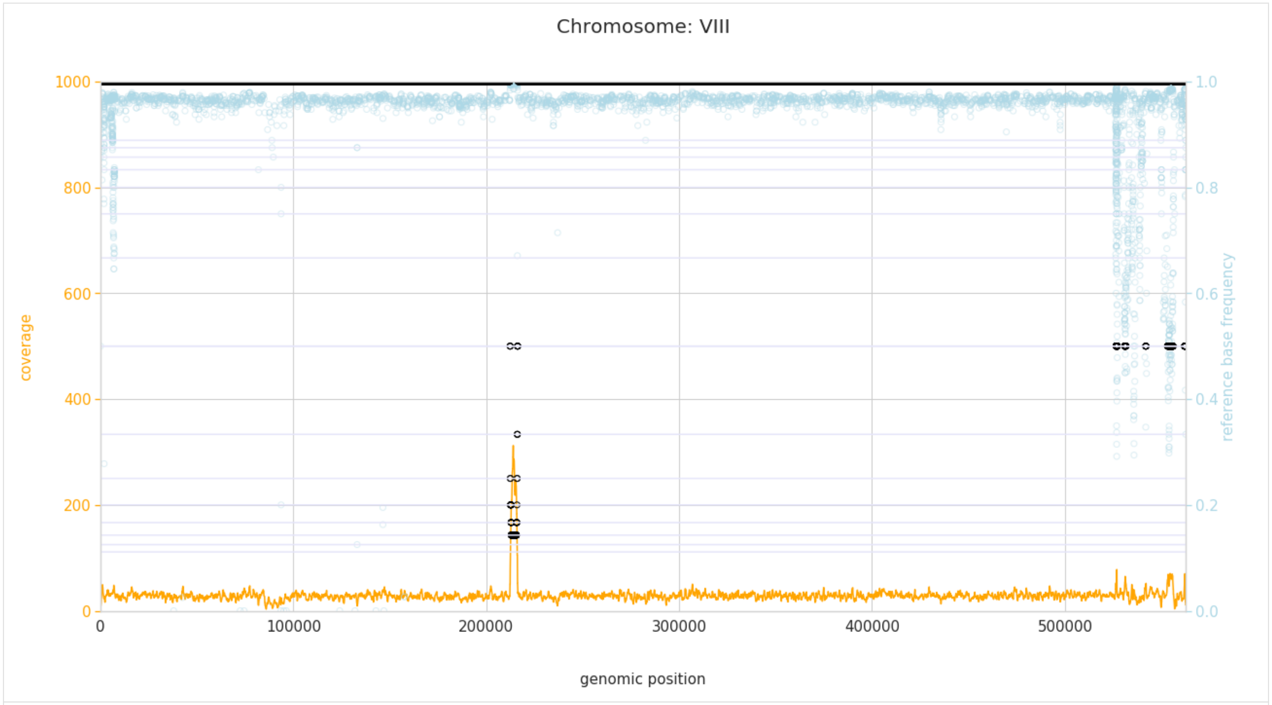

Chromosome: III
coverage
reference base frequency
genomic position
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
0
50000
100000
150000
200000
250000
300000

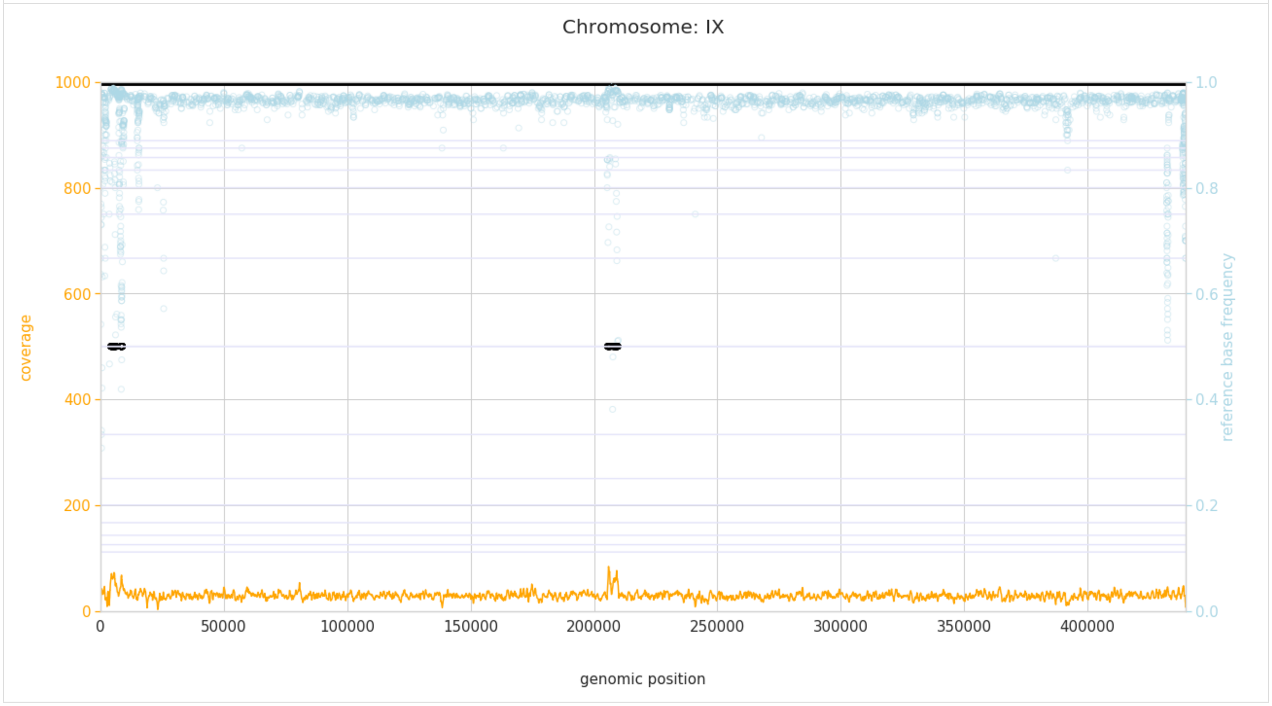

Chromosome: IV
coverage
reference base frequency
genomic position
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
200000
400000
600000
800000
1000000
1200000
1400000

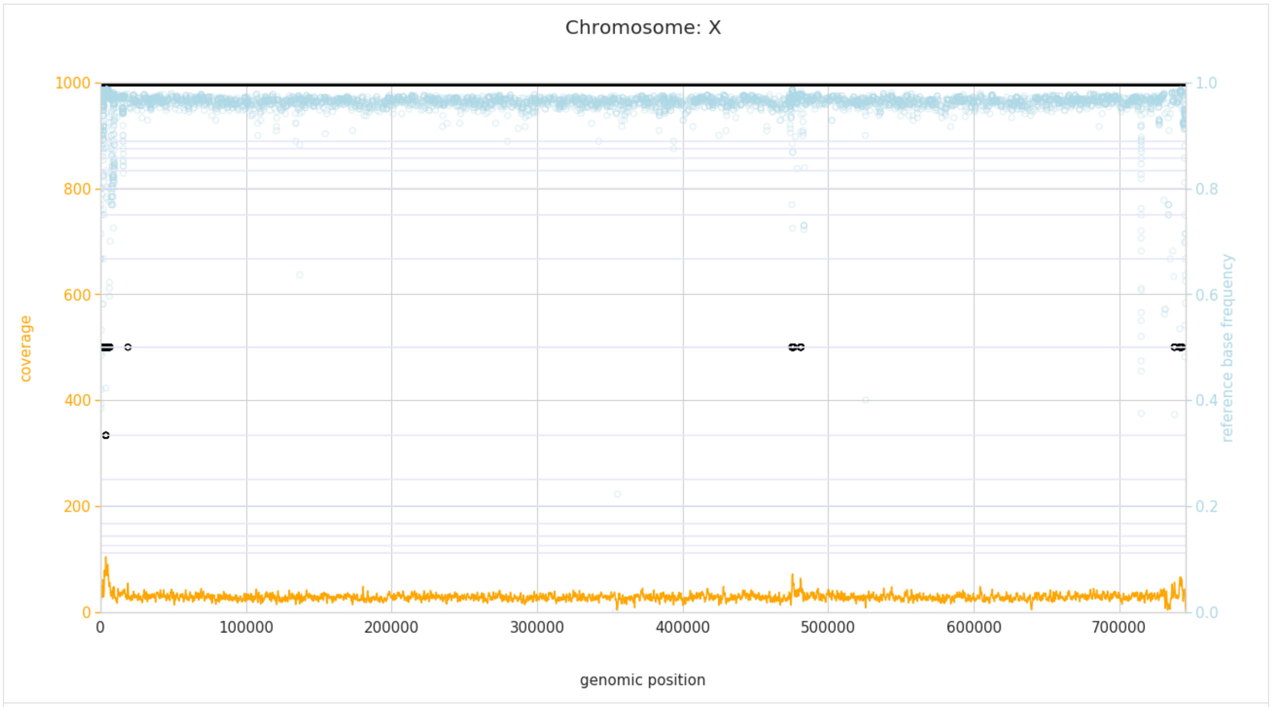

Chromosome: V
coverage
reference base frequency
genomic position
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
0
100000
200000
300000
400000
500000

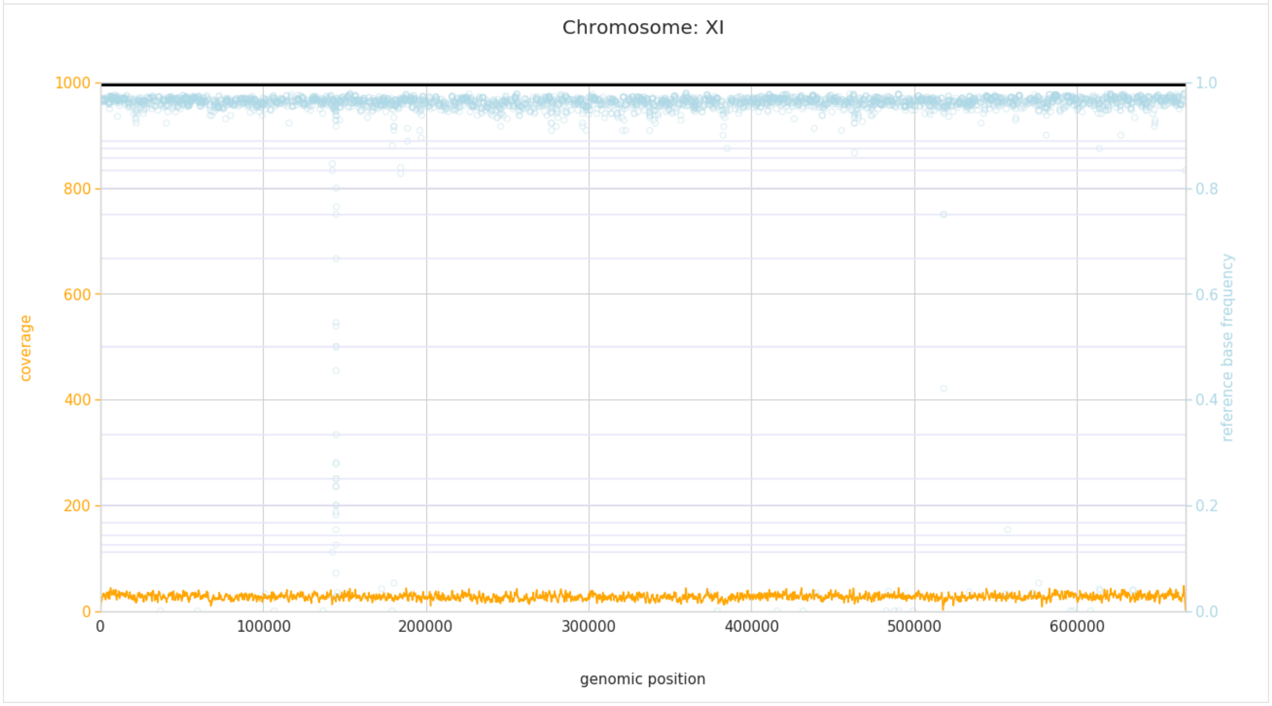

Chromosome: VI
coverage
reference base frequency
genomic position
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
0
50000
100000
150000
200000
250000

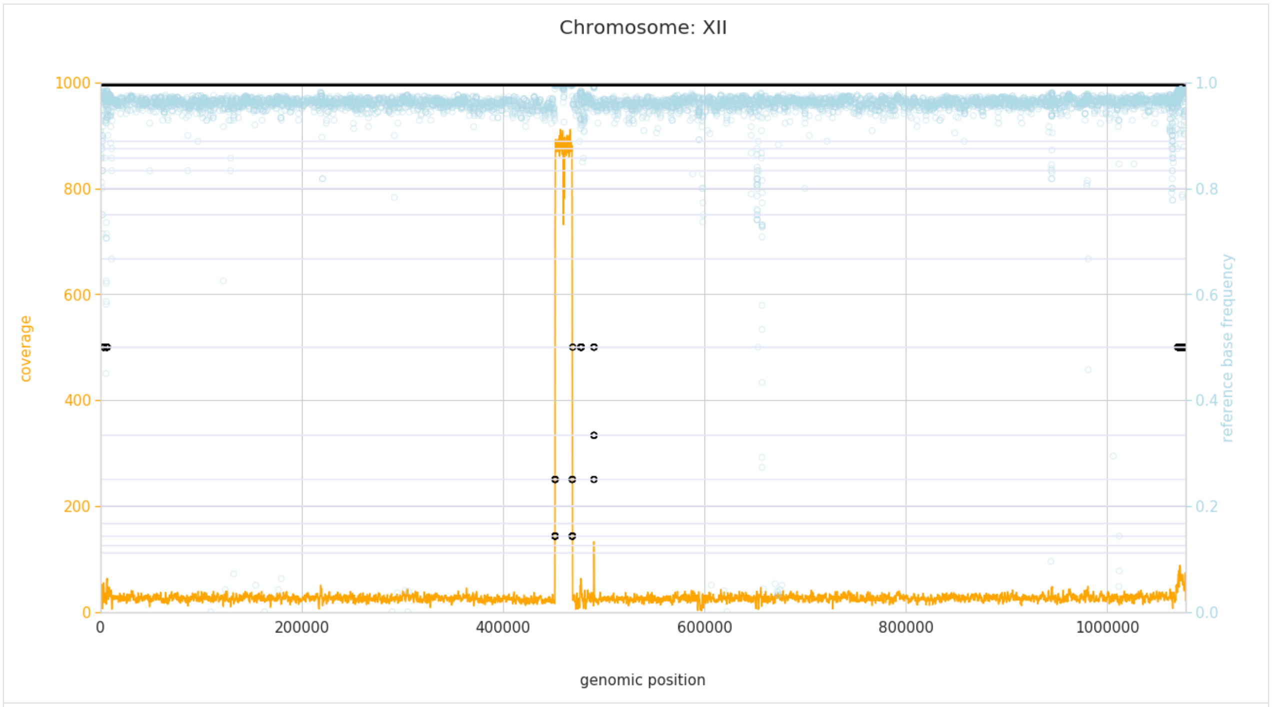

Chromosome: VII
coverage
reference base frequency
genomic position
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
0
200000
400000
600000
800000
1000000

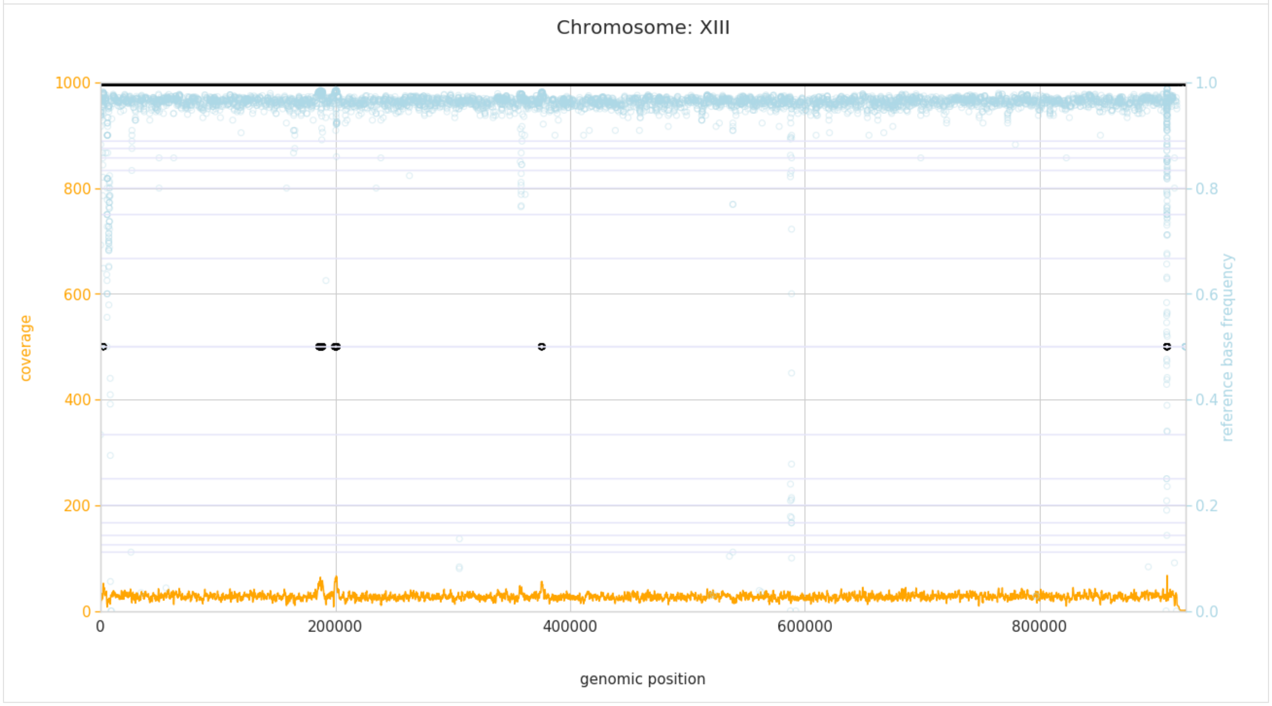

Chromosome: VIII
coverage
reference base frequency
genomic position
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
0
100000
200000
300000
400000
500000

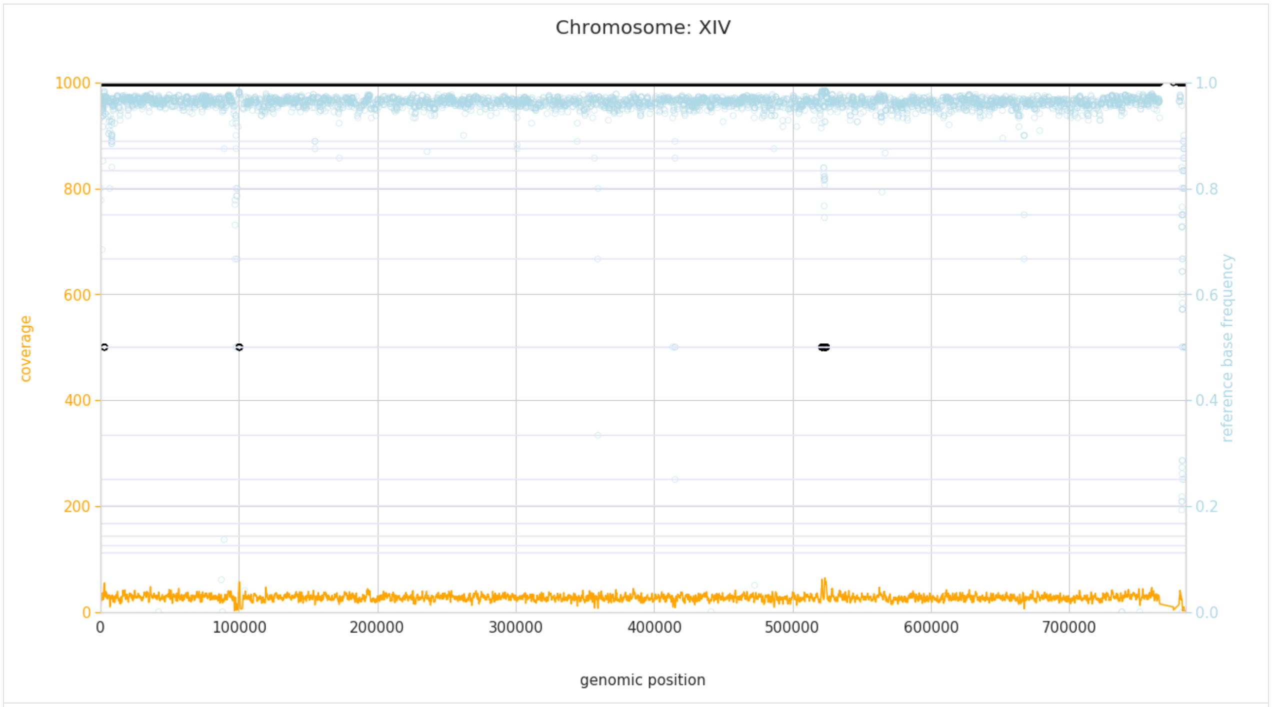

Chromosome: IX
coverage
reference base frequency
genomic position
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
0
50000
100000
150000
200000
250000
300000
350000
400000

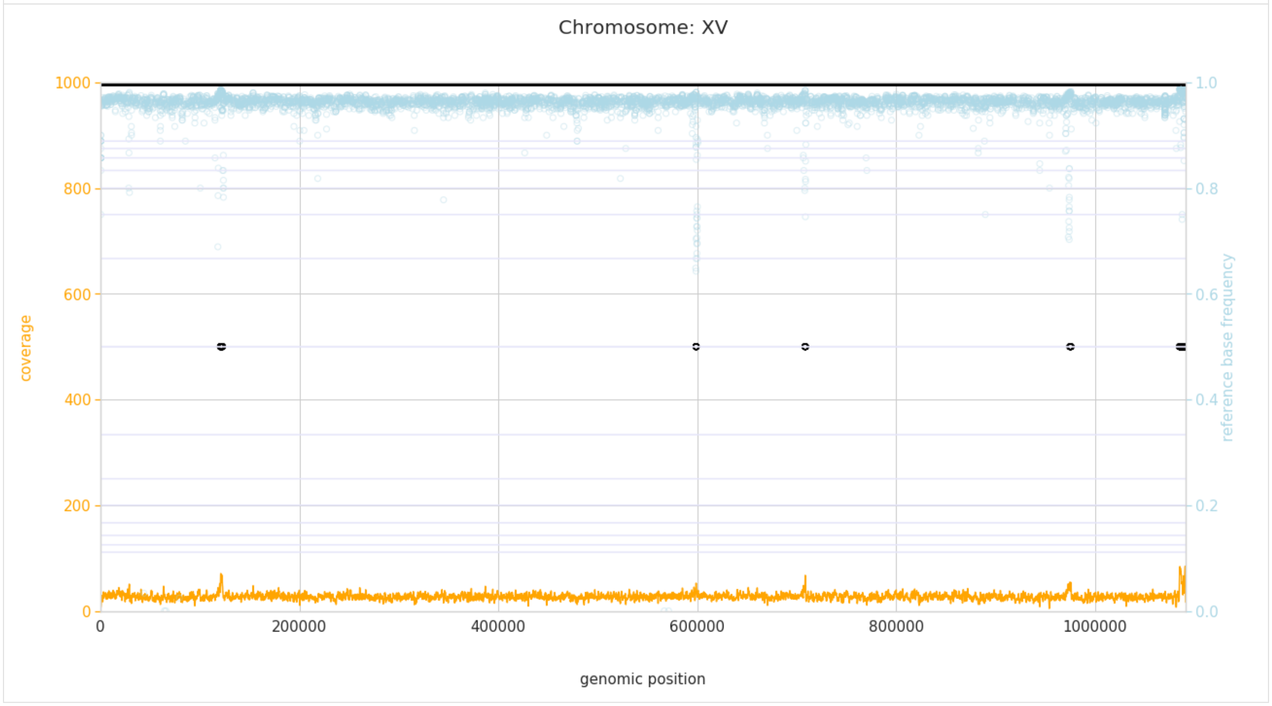

Chromosome: X
coverage
reference base frequency
genomic position
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
0
100000
200000
300000
400000
500000
600000
700000

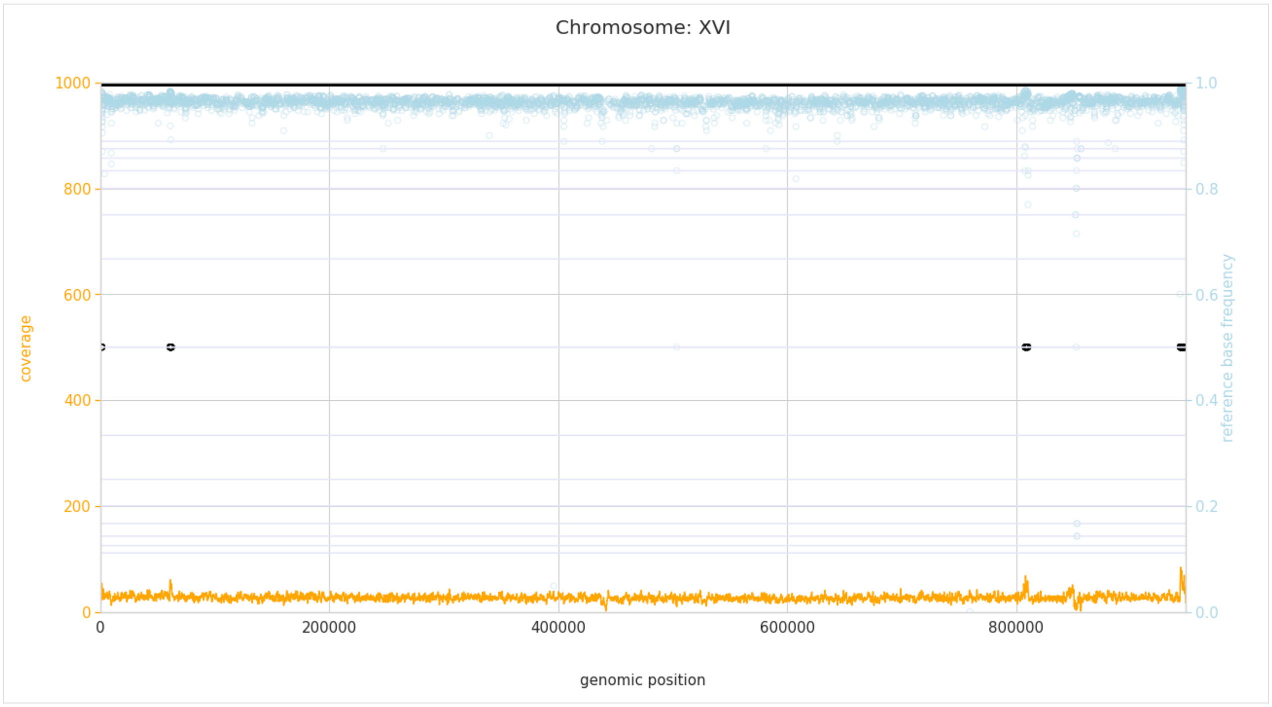

Chromosome: XI
coverage
reference base frequency
genomic position
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
0
100000
200000
300000
400000
500000
600000


Chromosome: XII
coverage
reference base frequency
genomic position
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
0
200000
400000
600000
800000
1000000


Chromosome: XIII
coverage
reference base frequency
genomic position
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
0
200000
400000
600000
800000


Chromosome: XIV
coverage
reference base frequency
genomic position
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
0
100000
200000
300000
400000
500000
600000
700000


Chromosome: XV
coverage
reference base frequency
genomic position
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
0
200000
400000
600000
800000
1000000



5.3 Comparing samples

A naive karyotype comparison of the samples is incorporated in isomut2py. However, we suggest using this option more as an exploratory step rather than a chosen tool for the accurate detection of copy number variations.

As a necessary first step, the ploidy estimation for another sample has to be performed. This can be easily achieved by slightly modifying the code used above:

```
[5]: PEparam_2 = im2.examples.load_example_ploidyest_settings(example_data_
↪path=exampleRawDataDir,
                                                              output_dir=examplePEResultsDir_
↪2)
```

```
[6]: PEst_2 = im2.ploidyestimation.PloidyEstimator(**PEparam_2)
PEst_2.bam_filename = 'B.bam'
```

```
[7]: PEst_2.run_ploidy_estimation(user_defined_hapcov=25)
```

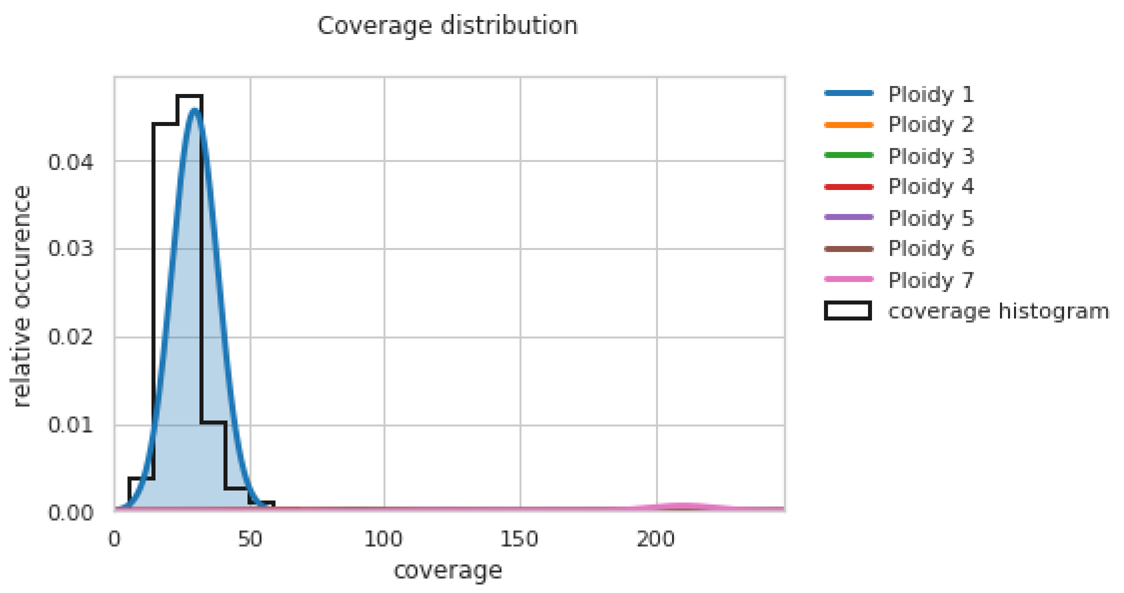
```
2018-11-15 14:35:37 - Ploidy estimation for file B.bam



        2018-11-15 14:35:37 - Preparing for parallelization...
                2018-11-15 14:35:37 - Defining parallel blocks ...
                2018-11-15 14:35:37 - Done

        2018-11-15 14:35:37 - (All output files will be written to ../Documents/
↪isomut2py_example_data/PE_example_results_20181114_B)

        2018-11-15 14:35:37 - Generating temporary files, number of blocks to run: 107
        2018-11-15 14:35:37 - Currently running:  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15␣
↪16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43␣
↪44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71␣
↪72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99␣
↪100 101 102 103 104 105 106 107
```

(continues on next page)

(continued from previous page)

```
        2018-11-15 14:36:06 - Temporary files created, merging, cleaning up...

        2018-11-15 14:36:06 - Collecting data for coverage distribution, using␣
↪user-defined haploid coverage (25)...

        2018-11-15 14:36:06 - Fitting equidistant Gaussians to the coverage␣
↪distribution using the raw haploid coverage as prior...

                2018-11-15 14:43:14 - Parameters of the distribution are saved to: ../
↪Documents/isomut2py_example_data/PE_example_results_20181114_B/GaussDistParams.pkl

                2018-11-15 14:43:14 - Final estimate for the haploid coverage: 29.
↪99889722813764

        2018-11-15 14:43:14 - Estimating local ploidy using the previously determined␣
↪Gaussians as priors on chromosomes:  I II III IV V VI VII VIII IX X XI XII XIII XIV␣
↪XV XVI

        2018-11-15 14:47:40 - Generating final bed file...


        2018-11-15 14:47:41 - Generating HTML report...


2018-11-15 14:48:13 - Ploidy estimation finished. (0 day(s), 0 hour(s), 12 min(s), 35␣
↪sec(s).)
```

[8]:
```
f = PEst_2.plot_coverage_distribution()
```



Now the comparison can be done in a single step with:

[13]:
```
PEst.compare_with_other(PEst_2)
```

```
2018-11-15 14:56:17 - Comparing regions of different ploidies in detail
```

(continues on next page)

(continued from previous page)

```
        2018-11-15 14:56:17 - Number of intervals to check: 2

                2018-11-15 14:56:17 - Currently running: 1 2

        2018-11-15 14:56:17 - Differing genomic intervals with quality scores saved␣
↪to file ../Documents/isomut2py_example_data/PE_example_results_20181114A_VS_B_COMP.
↪bed


2018-11-15 14:56:17 - Comparison finished.
```

The comparison results stored in the above file can be checked with:

```
[17]: !cat ../Documents/isomut2py_example_data/PE_example_results_20181114/A_VS_B_COMP.bed
```

```
chrom   intStart        intEnd  ploidy1 ploidy2 LOH1    LOH2    intLength      ␣
↪quality
VII     289734  291768  1       3       0       0       2034    619987021.8068602
XII     1072935 1075050 2       1       0       0       2115    0.8403883817503438
```

5.4 Correcting bad fits, loading fitted coverage distributions

As the distribution fitting is the most time consuming part of the ploidy estimation, it can be useful to load an already fitted distribution that we can work with. The above used ploidy estimation pipeline automatically saves the parameters of the fitted distribution to a file in the output directory, so these can be easily reused.

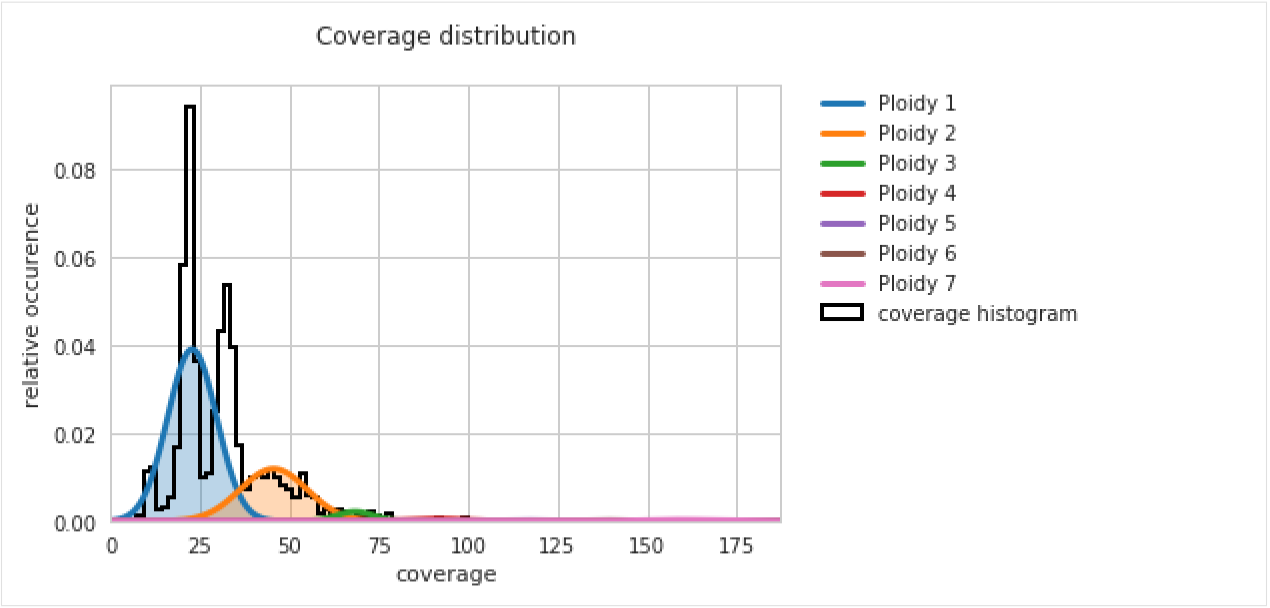
As an example, we will be using the already prepared files in the example dataset. (For download instructions, see *Getting started*.)

```
[3]: PEparam_3 = im2.examples.load_preprocessed_example_ploidyest_settings(example_data_
↪path=exampleDataDir)
```

```
[4]: PEst_3 = im2.ploidyestimation.PloidyEstimator(**PEparam_3)
```

```
[6]: PEst_3.load_cov_distribution_parameters_from_file(PEst_3.output_dir +
↪'GaussDistParams_incorrect.pkl')
```

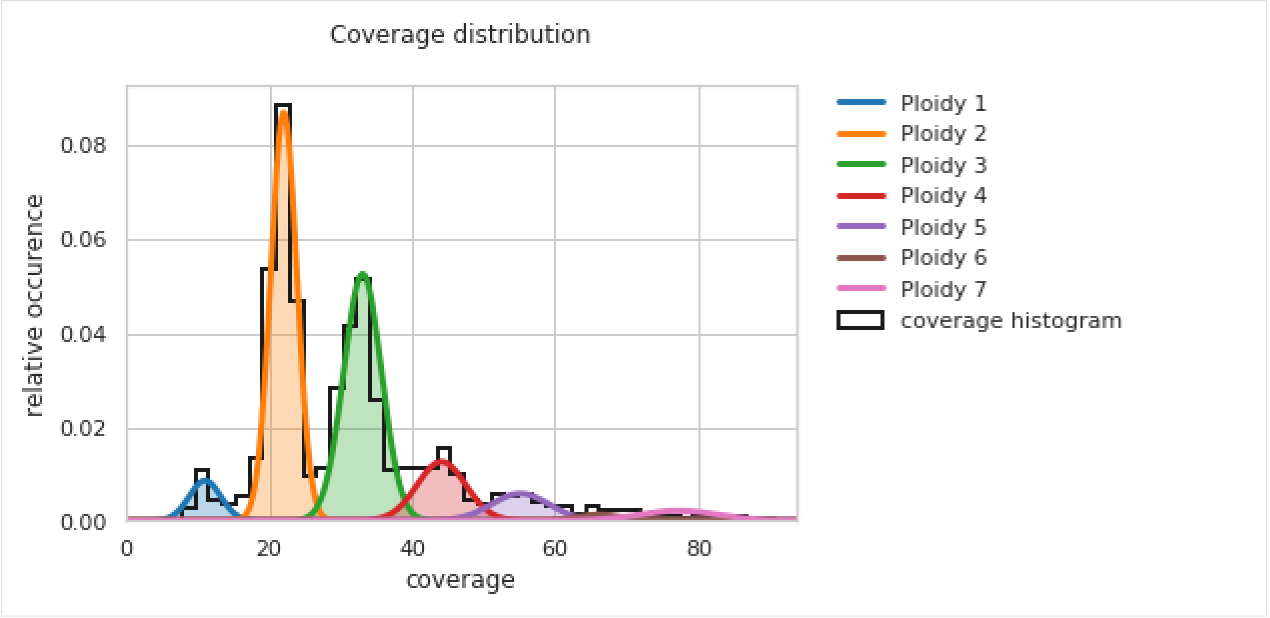
```
[7]: f = PEst_3.plot_coverage_distribution()
```



This is obviously a highly incorrect fit for this data. In cases like this, the above described procedure can be followed: the haploid coverage should be estimated manually (essentially by looking at the position of the first peak of the above histogram), and the fitting rerun with this user-defined value:

```
[9]: PEst_3.fit_gaussians(estimated_hapcov = 11)
```

```
[32]: f = PEst_3.plot_coverage_distribution()
```

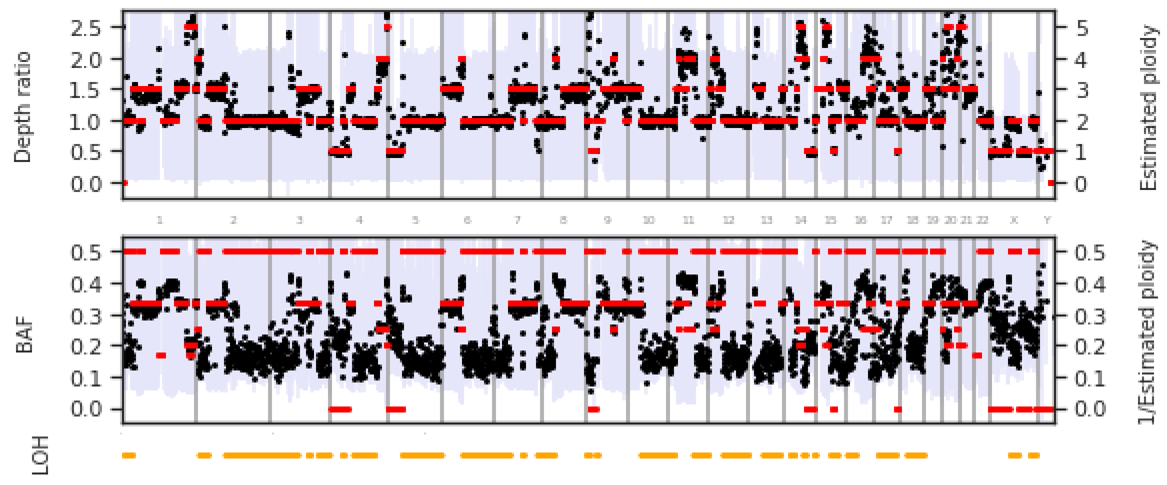


And finally, ploidy estimation can be rerun with the now correct prior distribution with:

```
[6]: PEst_3.PE_on_whole_genome()
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 X Y
```

```
[8]: f = PEst_3.plot_karyotype_summary()
```



5.5 Using estimated ploidies as input for mutation detection

Once the ploidy estimation has been performed as described, we can use this information to fine-tune mutation detection. Mutation calling is run on diploid genomes by default, however, when analysing non-diploid genomes or genomes with largely varying ploidies (as shown above), it can be useful to determine the ploidy more precisely. (This is important because the alternate allele frequency in mutated positions largely depends on the local ploidy.)

If the analysed organism (all of the samples in the sample set) has a constant ploidy different from 2 (for example, a fully haploid genome), its value can be simply set with:

```
mutDet = im2.mutationcalling.MutationCaller()
mutDet.constant_ploidy = 1
```

(For basic mutation calling, see *Getting started*.)

However, in cases when some of the analysed samples have varying ploidy along the length of the genome (for example, aneuploid cell lines), the local ploidy can be uniquely defined for each sample and each genomic position in a bed file.

As a best practice, we suggest performing the above ploidy estimation on the starting clones in a sample set. This creates a separate ploidy bed file for each sample group derived from the starting clones. These can be assigned to specific samples with (make sure to provide valid paths to the bed files):

```
[8]: im2.format.generate_ploidy_info_file(sample_groups=[['S1.bam', 'S2.bam', 'S3.bam'],
                                                          ['S4.bam', 'S5.bam', 'S6.bam',
     ↪'S7.bam']],
                                        group_bed_filepaths=['path_to_S1_ploidy.bed',
     ↪'path_to_S4_ploidy.bed'],
                                        filename='ploidy_info_file_for_test_set.txt')
```

The above function generates a ploidy info file to the path specified by `filename`:

```
[9]: !cat ploidy_info_file_for_test_set.txt
```

```
#file_path    sample_names_list
path_to_S1_ploidy.bed_im2format     S1.bam, S2.bam, S3.bam
path_to_S4_ploidy.bed_im2format     S4.bam, S5.bam, S6.bam, S7.bam
```

Now this file can be supplied to the mutation detection object with:

```
mutDet.ploidy_info_file = "ploidy_info_file_for_test_set.txt"
```

(Make sure to set the complete path to the file.)

For all samples in the dataset that are not included in the ploidy info file, the constant ploidy set with the `constant_ploidy` attribute will be used (default: 2). This is also true for any genomic regions that were not specified in the bed files.

CHAPTER 6

Further analysis, visualization

Once a basic set of mutations has been recovered for the analysed sample set, one can proceed with additional visualization and postprocessing steps. These are demonstrated here.

6.1 Recovering mutations

There are three options to produce a dataframe which stores all the mutations present in the investigated sample set:

1. Run the `isomut2py` mutation detection pipeline. (See *Getting started*.)
2. Load the list of mutations previously detected by `isomut2py`.
3. Import mutations detected by external tools. (See *Importing external mutations*.)

For now, we will use the second option with a dataframe provided in the example dataset. (For instructions on download, see *Getting started*.) The reference genome used for this set of mutations can be found in the same directory.

```
[73]: import pandas as pd
      import isomut2py as im2
      %matplotlib inline
```

```
[74]: exampleDataDir='/nagyvinyok/adat83/sotejedlik/orsi/'
```

```
[81]: mutations_df = pd.read_csv(exampleDataDir+'isomut2py_example_dataset/Vizualisation/
       ↪mutations_dataframe.csv', sep='\t')
```

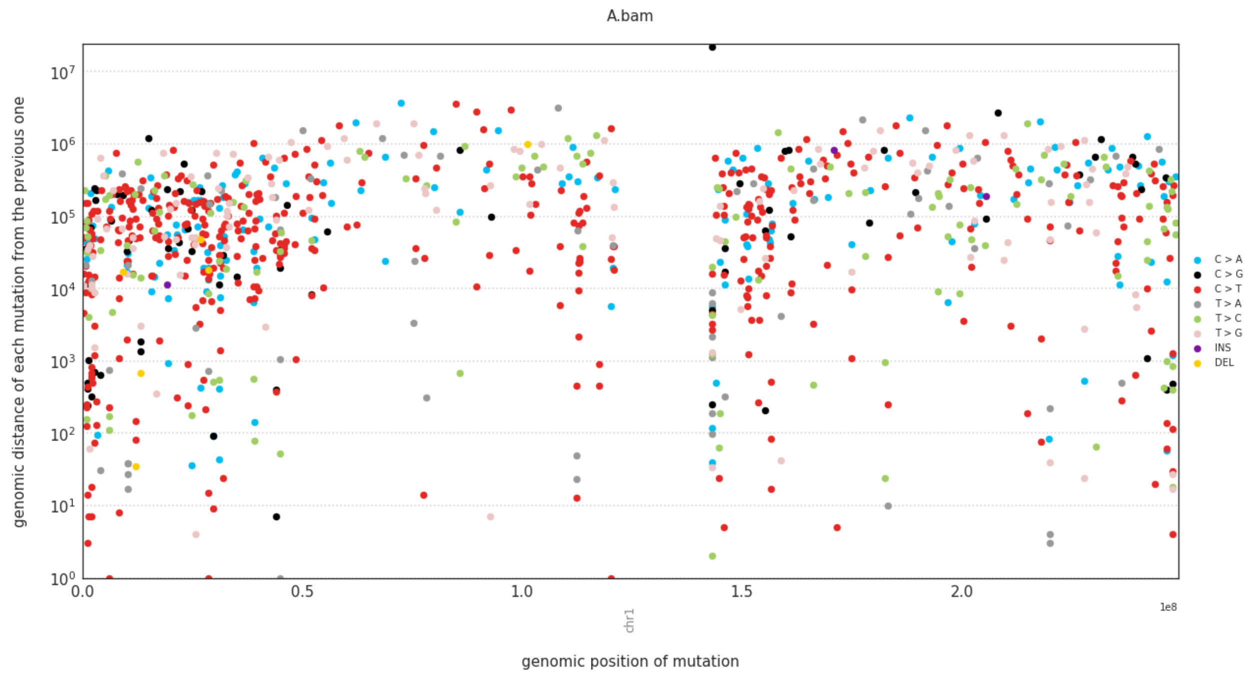
6.2 Plotting mutations on rainfall plots

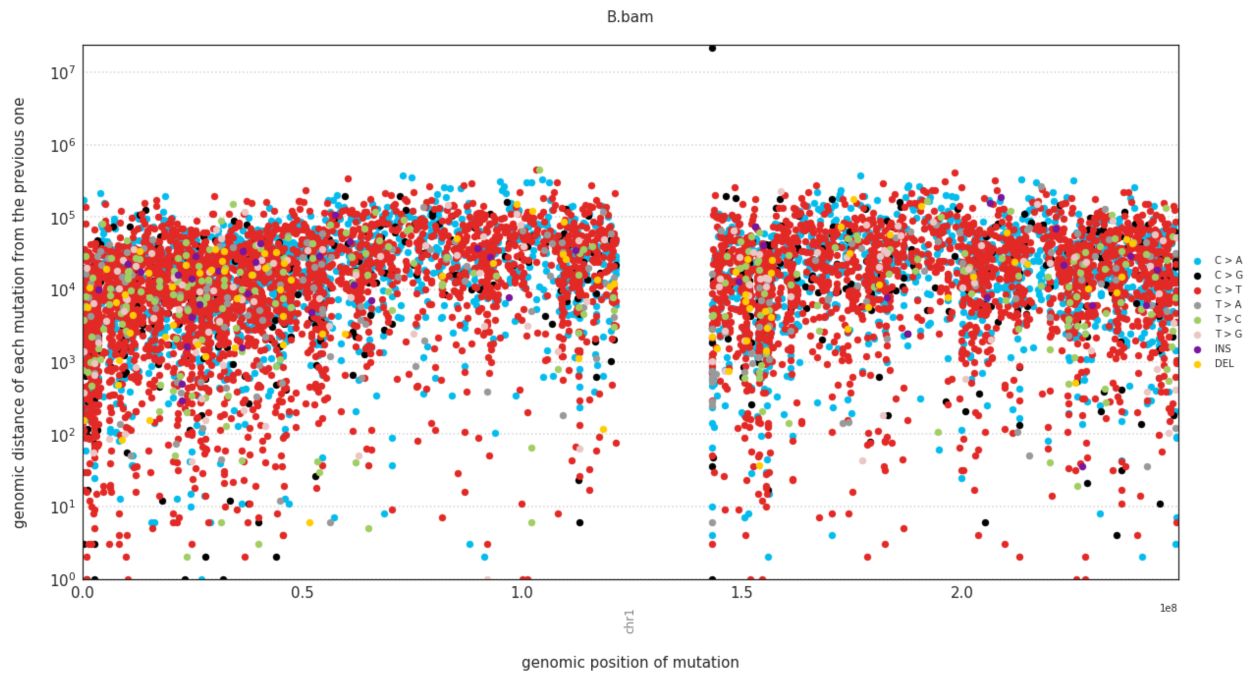
(To simply plot the number of mutations in each sample, see *Getting started*.)

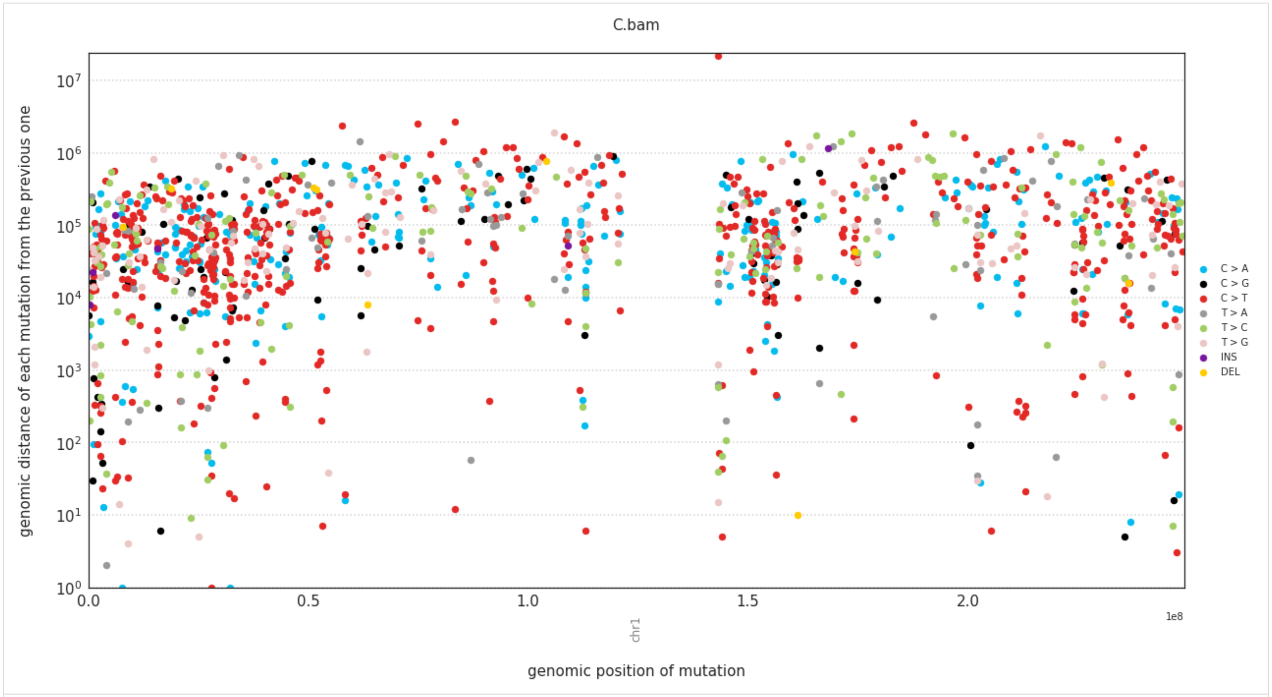
In a rainfall plot, the horizontal axis represents the genomic position of a mutations, while the vertical axis shows the distance from the previous mutation. Thus mutational clusters appear on the bottom part of the rainfall plots as a cluster of dots.

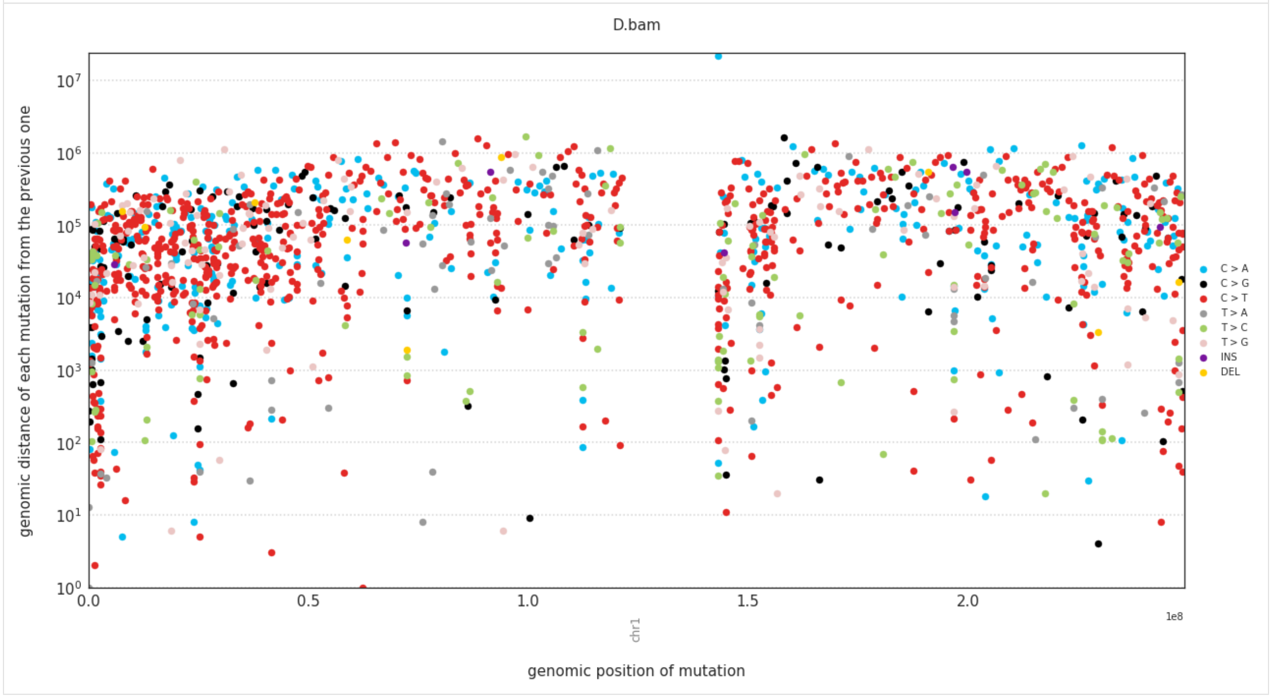
These can be plotted with:

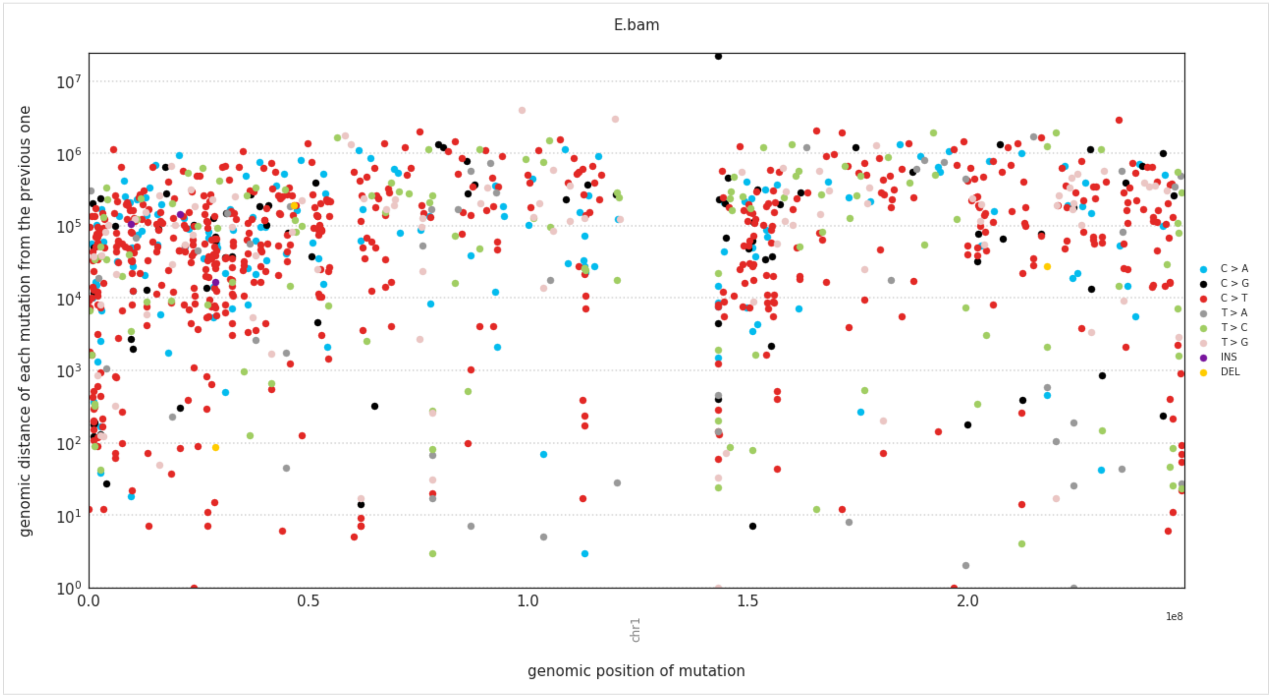
```
[86]: f = im2.plot.plot_rainfall(mutations_dataframe=mutations_df,
                                   ref_fasta=exampleDataDir+'isomut2py_example_dataset/
      ↪Vizualisation/chr1.fa')
```






C.bam
genomic distance of each mutation from the previous one
genomic position of mutation
chr1
1e8
C > A
C > G
C > T
T > A
T > C
T > G
INS
DEL

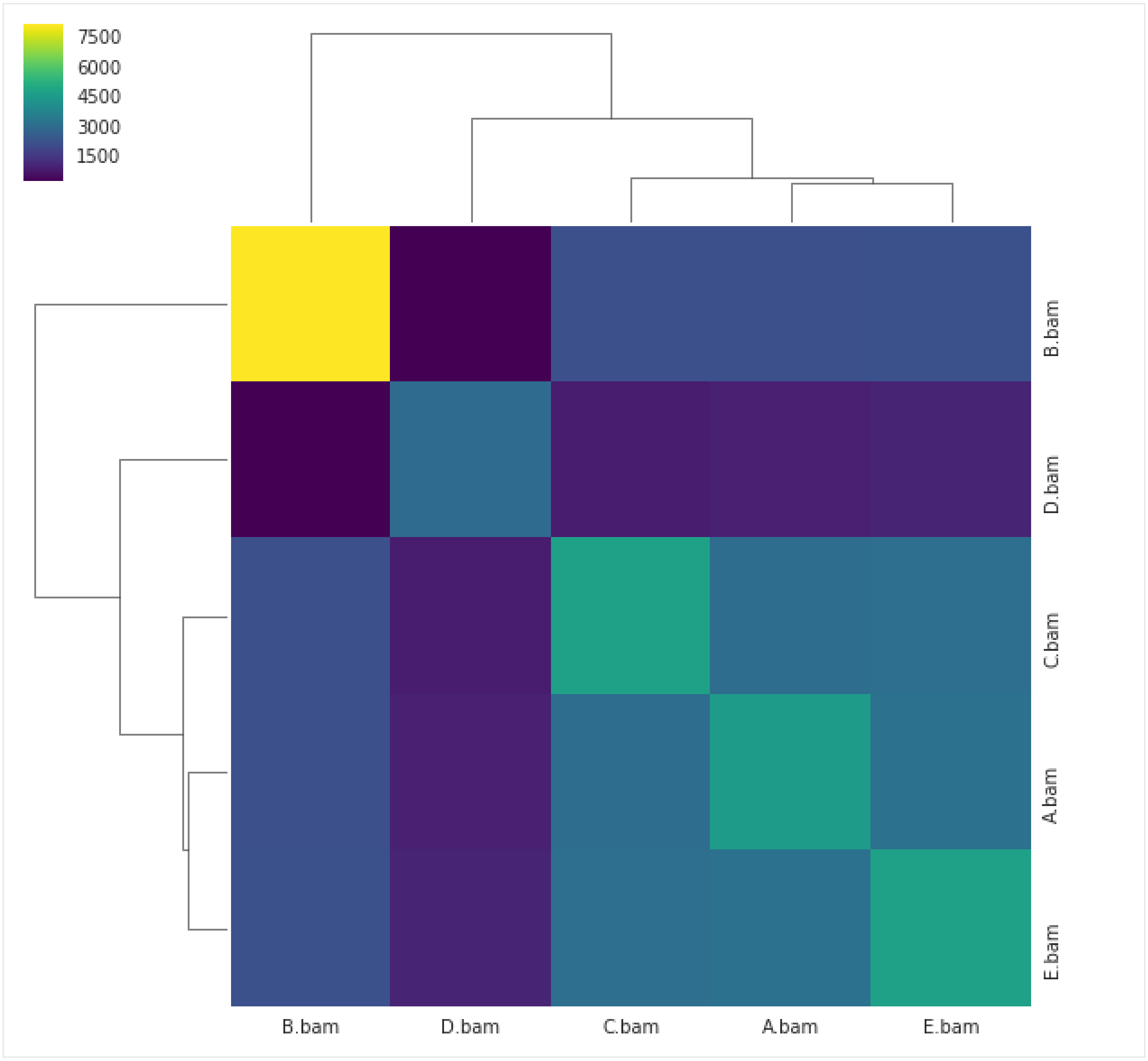

D.bam
genomic distance of each mutation from the previous one
genomic position of mutation
chr1
1e8
C > A
C > G
C > T
T > A
T > C
T > G
INS
DEL



6.3 Plotting a simple hierarchical clustering of the samples

A naive hierarchical clustering of the analysed samples based on the number of shared mutations can be plotted with:

```
[88]: f = im2.plot.plot_hierarchical_clustering(mutations_dataframe=mutations_df)
```



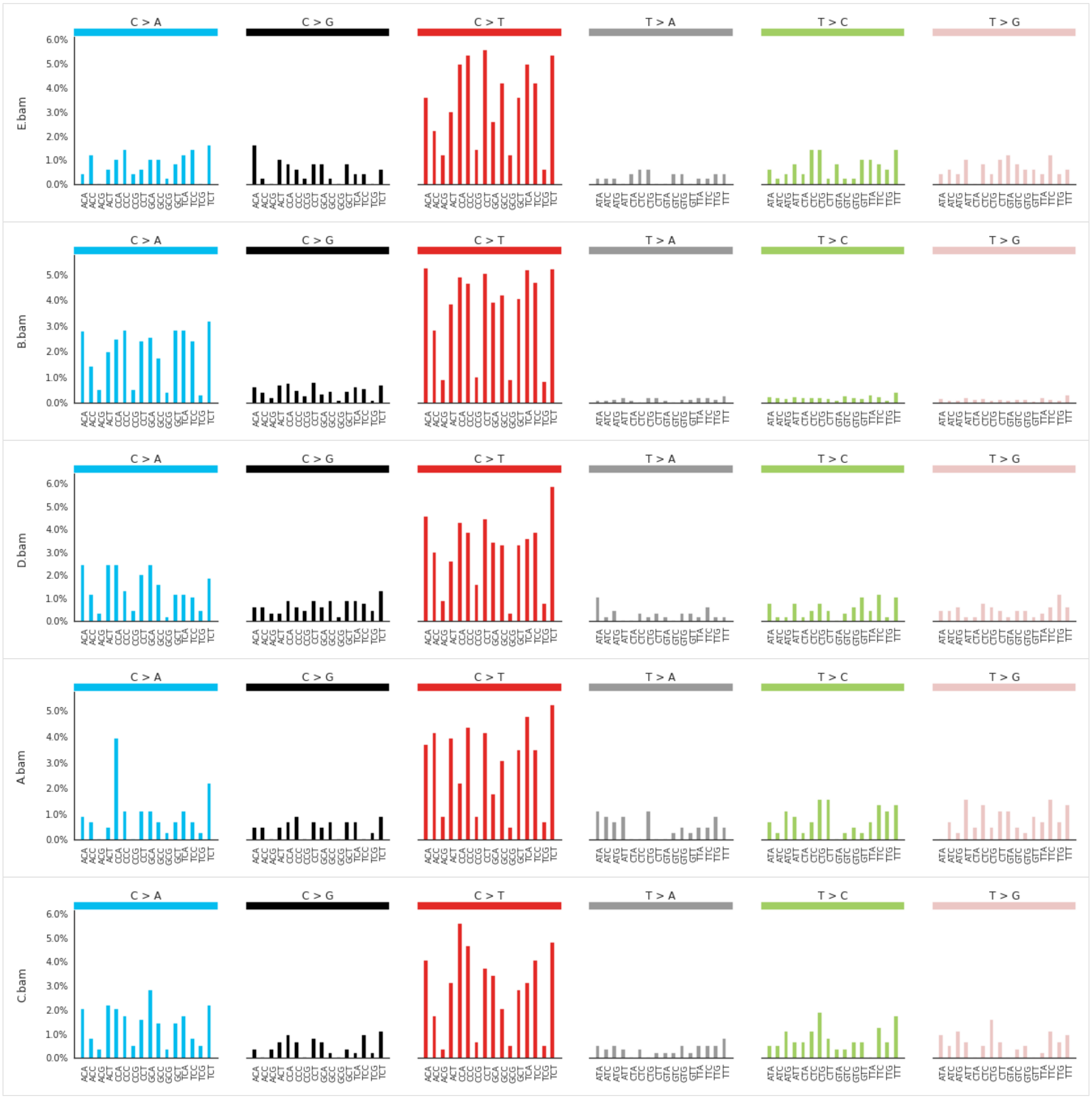
6.4 Plotting the spectra of mutations

The spectra of different mutation types are defined in this paper.

To calculate and plot the spectra of SNVs in the samples, use:

```
[83]: SNV_spectra = im2.postprocess.calculate_SNV_spectrum(ref_fasta=exampleDataDir+
      ↪'isomut2py_example_dataset/Vizualisation/chr1.fa',
                                                           mutations_dataframe=mutations_df)
```

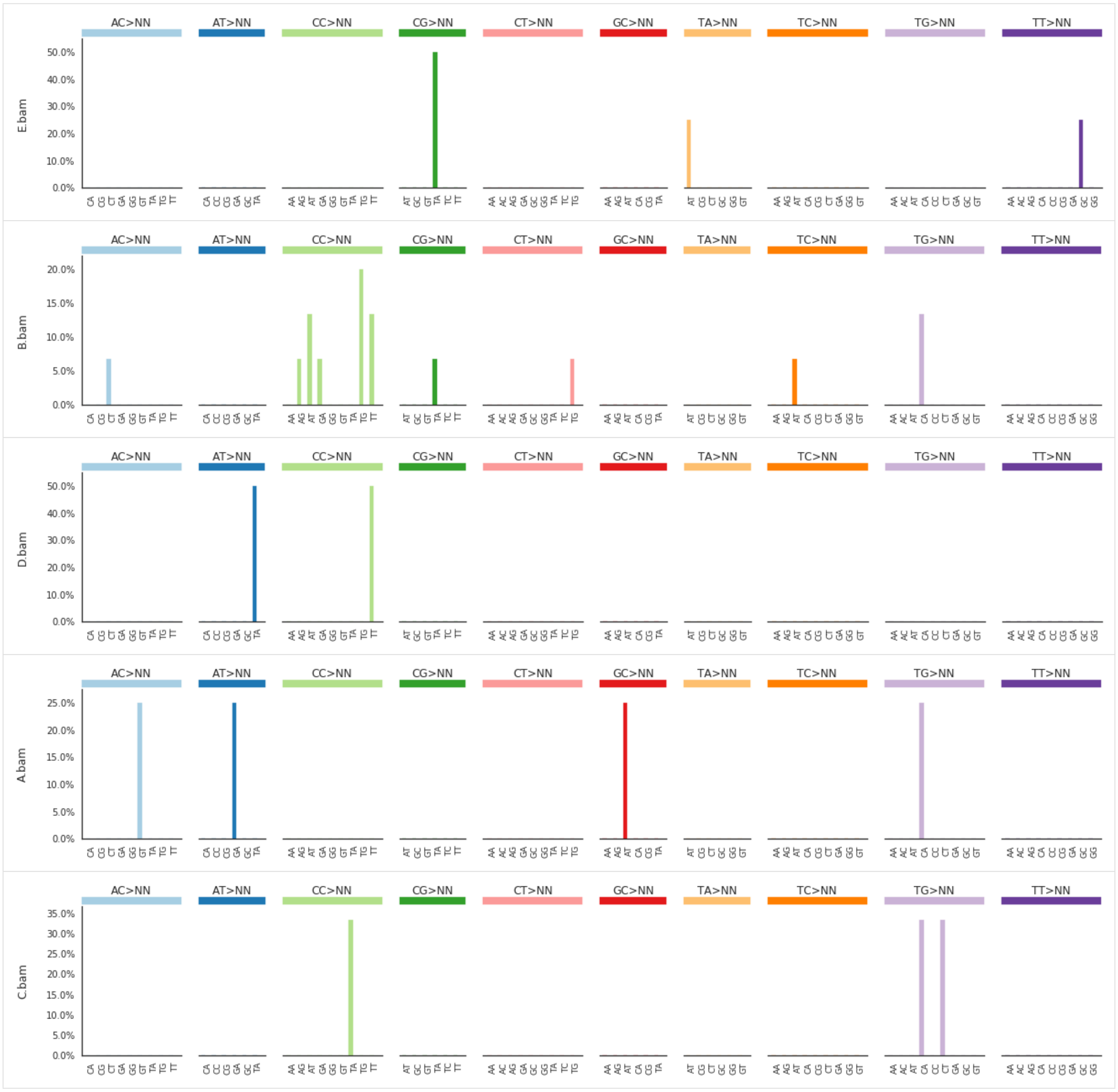
```
[85]: f = im2.plot.plot_SNV_spectrum(spectrumDict=SNV_spectra, normalize_to_1=True)
```



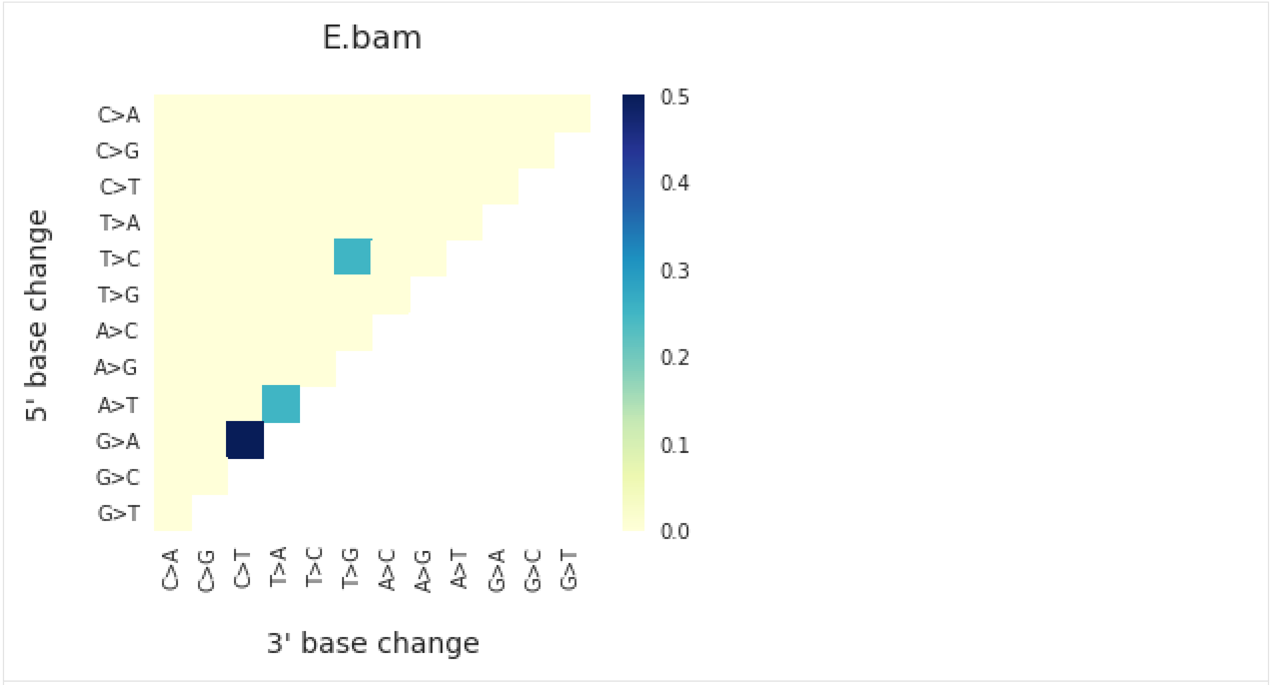
To get the DNV spectra in a linear and a heatmap format, use:

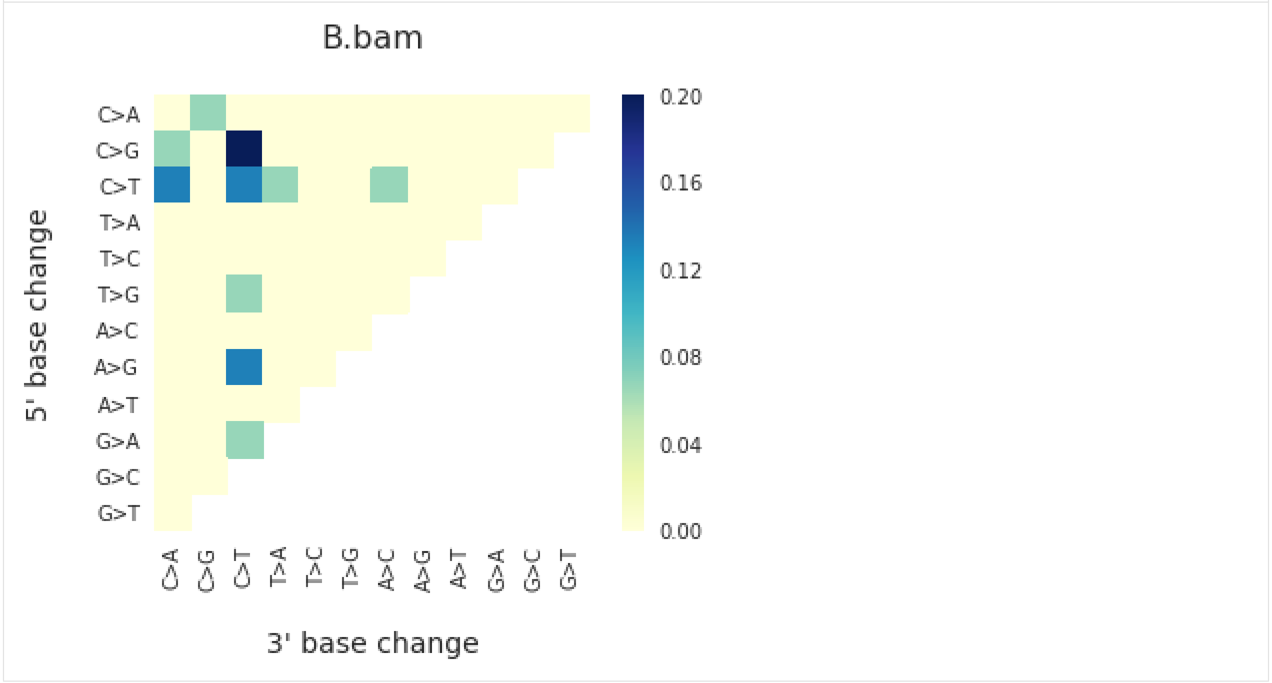
```
[89]: DNVspectra = im2.postprocess.calculate_DNV_spectrum(mutations_dataframe=mutations_df)
      DNVmatrix = im2.postprocess.calculate_DNV_matrix(mutations_dataframe=mutations_df)
```

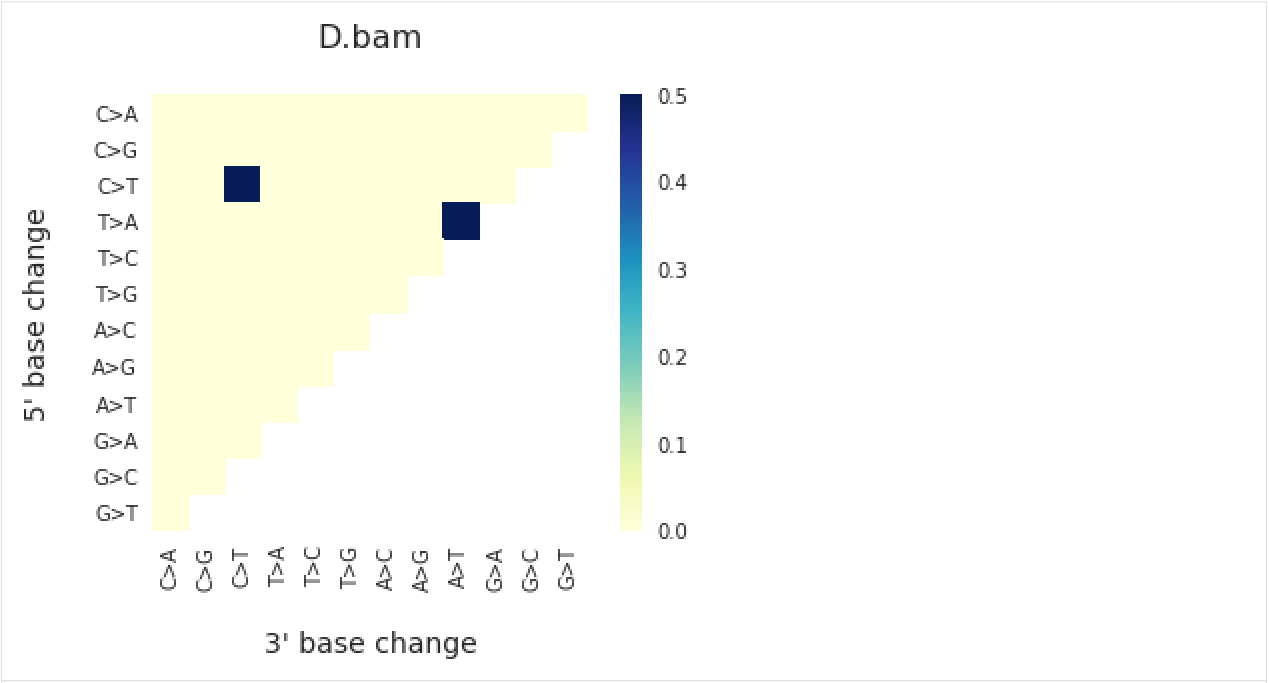
```
[90]: f = im2.plot.plot_DNV_spectrum(spectrumDict=DNVspectra, normalize_to_1=True)
```

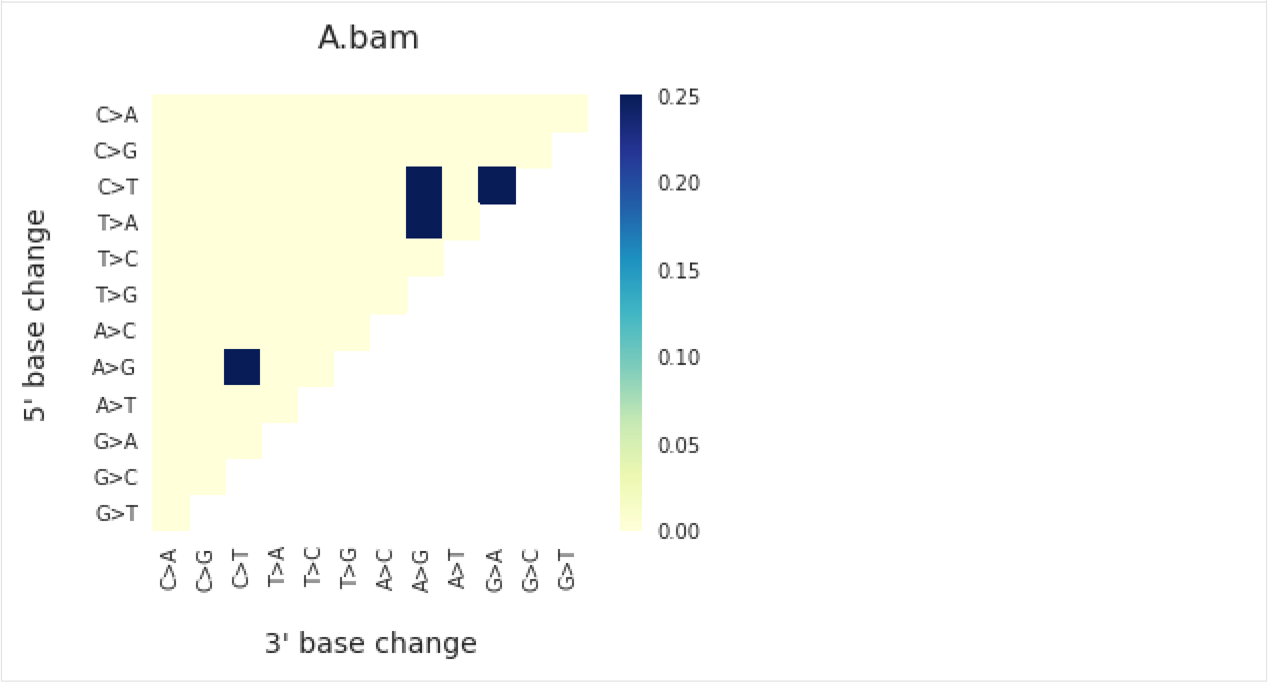


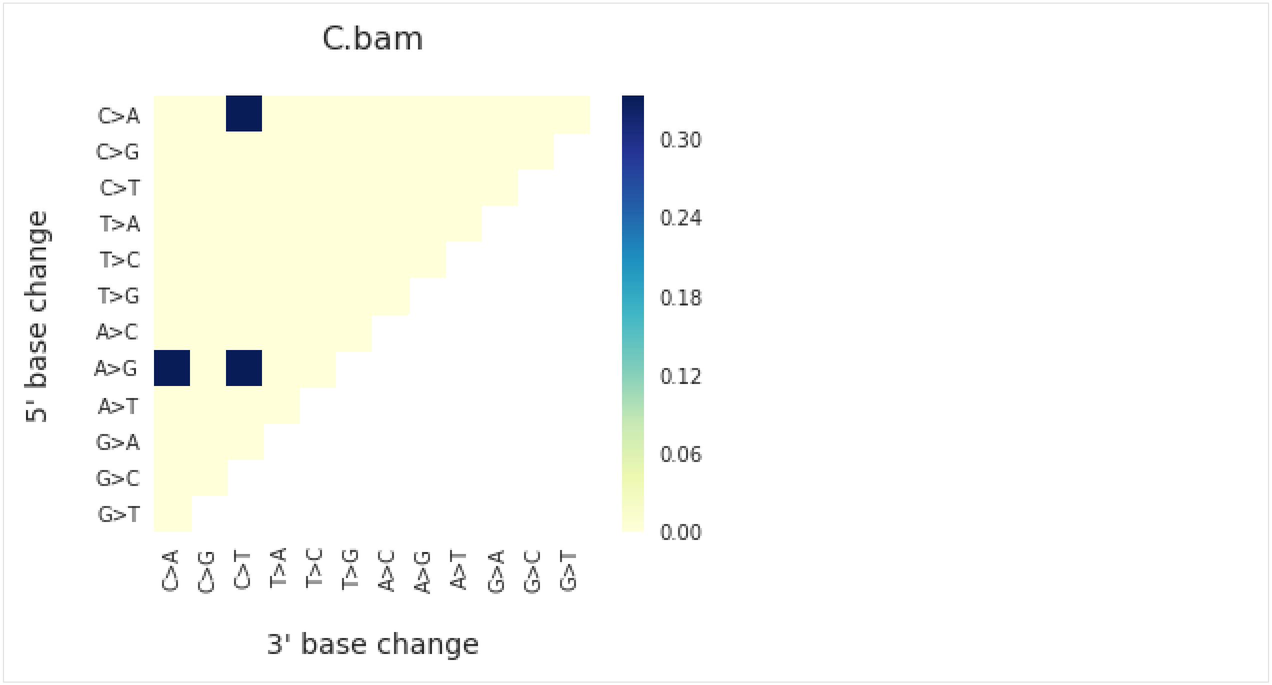
```
[91]: f = im2.plot.plot_DNV_heatmap(matrixDict=DNVmatrix, normalize_to_1=True)
```


E.bam
5' base change
3' base change
C>A
C>G
C>T
T>A
T>C
T>G
A>C
A>G
A>T
G>A
G>C
G>T
0.5
0.4
0.3
0.2
0.1
0.0


B.bam
5' base change
3' base change
C>A
C>G
C>T
T>A
T>C
T>G
A>C
A>G
A>T
G>A
G>C
G>T
0.20
0.16
0.12
0.08
0.04
0.00


D.bam
5' base change
C>A
C>G
C>T
T>A
T>C
T>G
A>C
A>G
A>T
G>A
G>C
G>T
3' base change
0.5
0.4
0.3
0.2
0.1
0.0

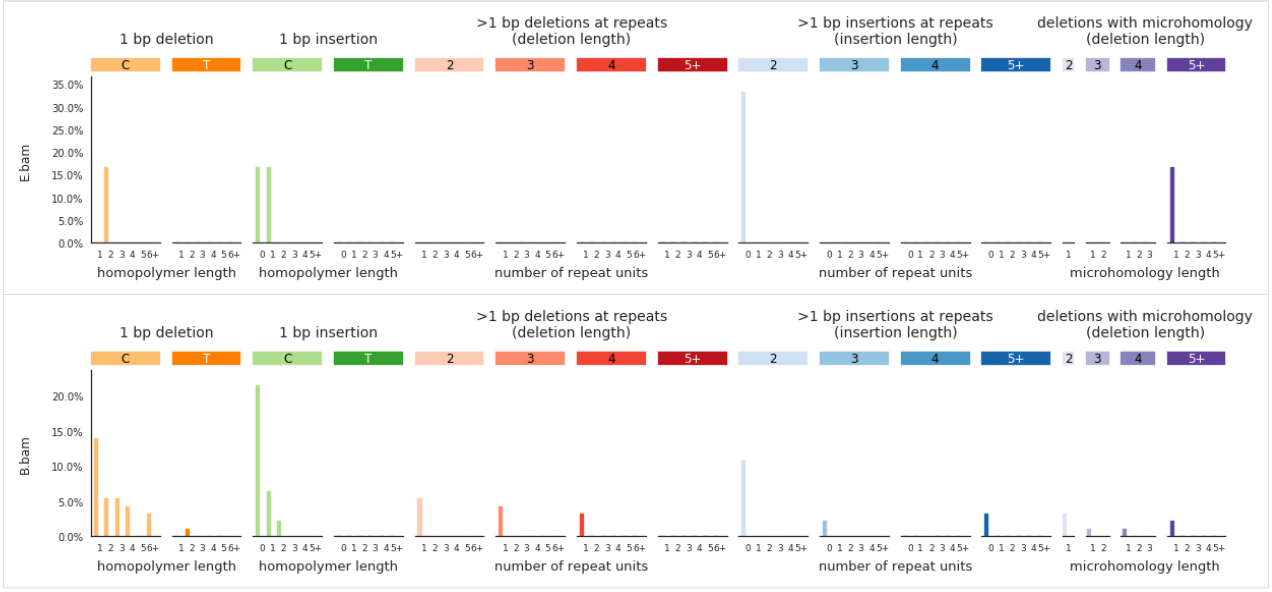

A.bam
5' base change
C>A
C>G
C>T
T>A
T>C
T>G
A>C
A>G
A>T
G>A
G>C
G>T
3' base change
0.25
0.20
0.15
0.10
0.05
0.00

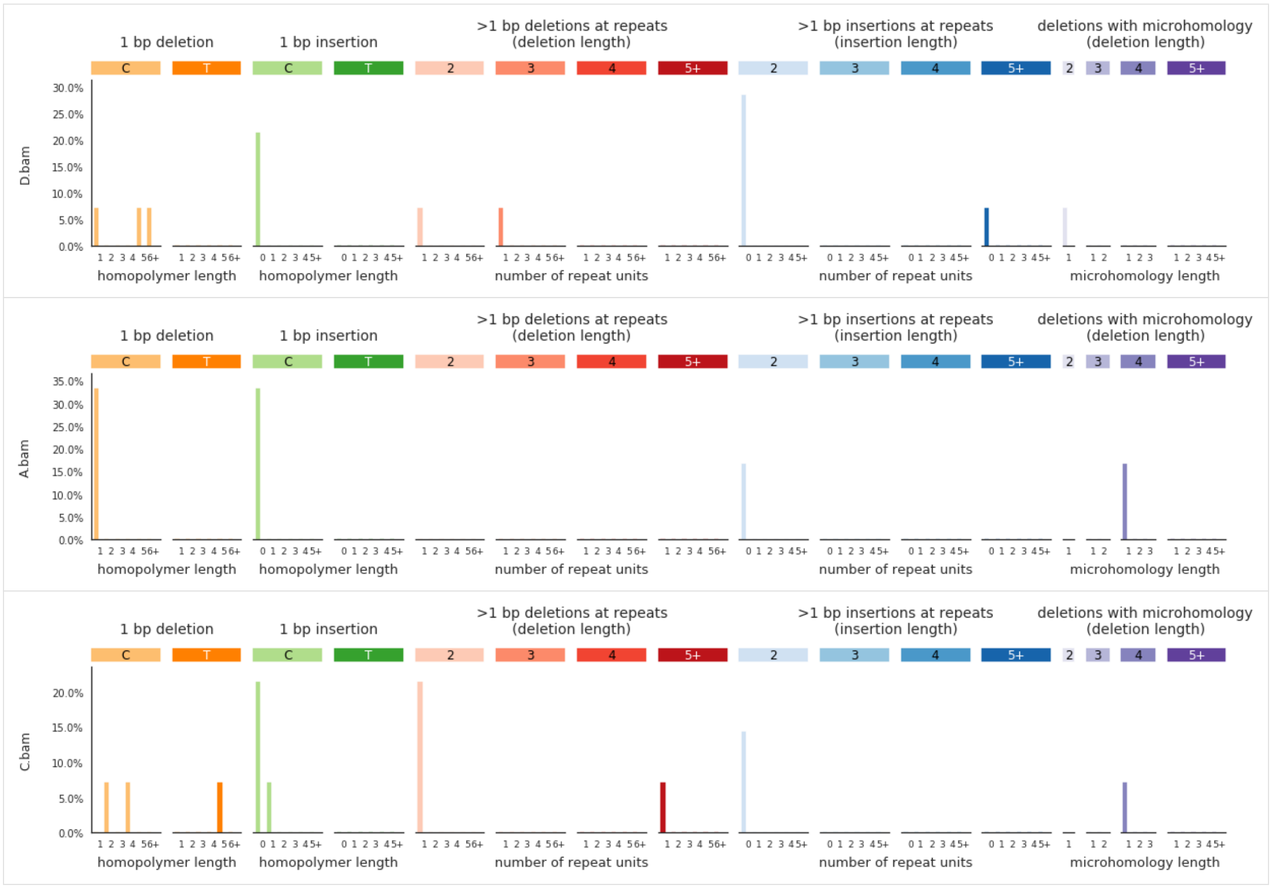


And finally, to get indel spectra, use:

```
[92]: IDspecra = im2.postprocess.calculate_indel_spectrum(ref_fasta=exampleDataDir+
↪'isomut2py_example_dataset/Vizualisation/chr1.fa',
                                                    mutations_dataframe=mutations_df)
```

```
[93]: f = im2.plot.plot_indel_spectrum(spectrumDict=IDspecra, normalize_to_1=True)
```

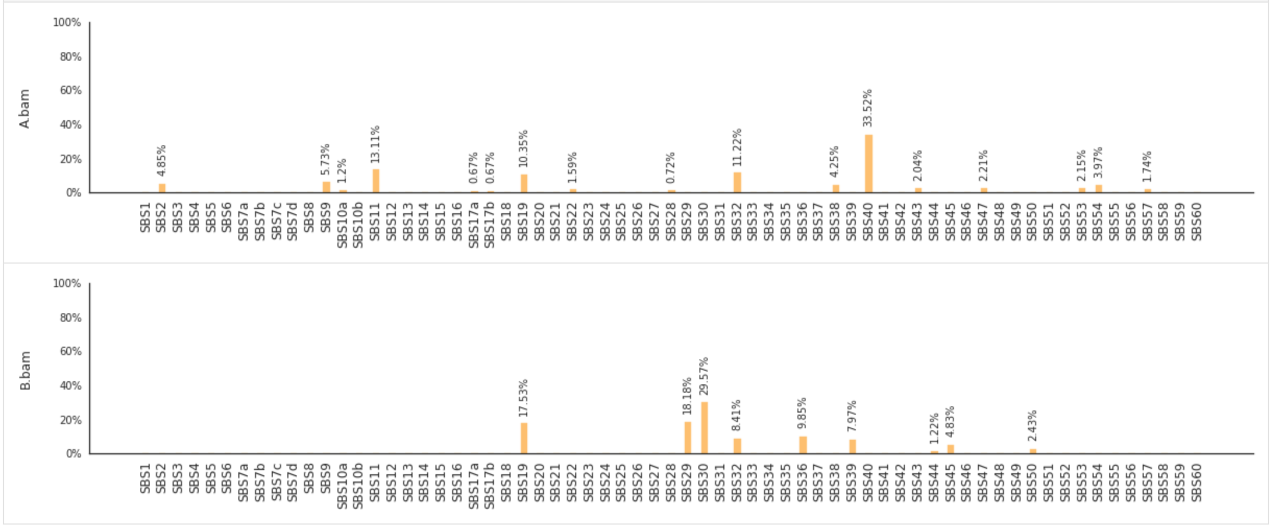


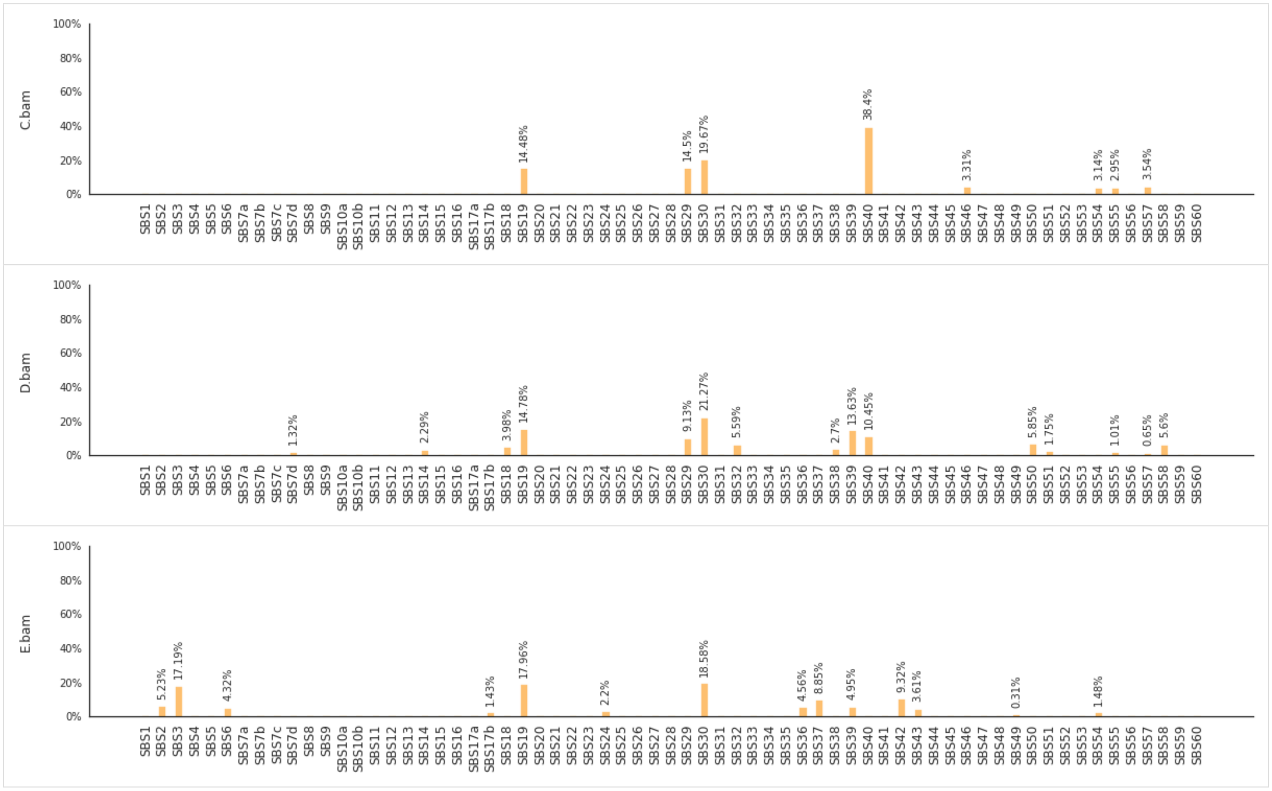


6.5 Decomposing mutational spectra to reference signatures

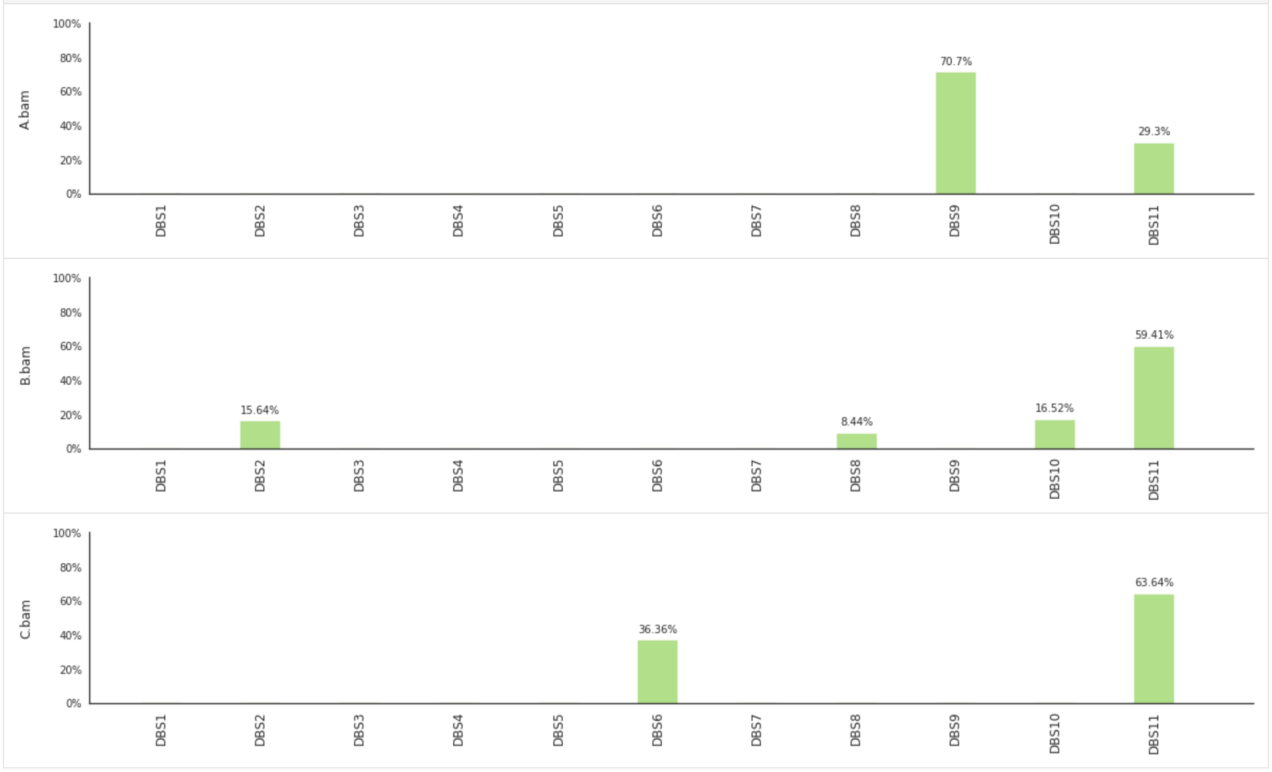
Using the consensus signatures described in the paper above, we can decompose mutational spectra to weighted contributions of the reference signatures. These can be achieved with:

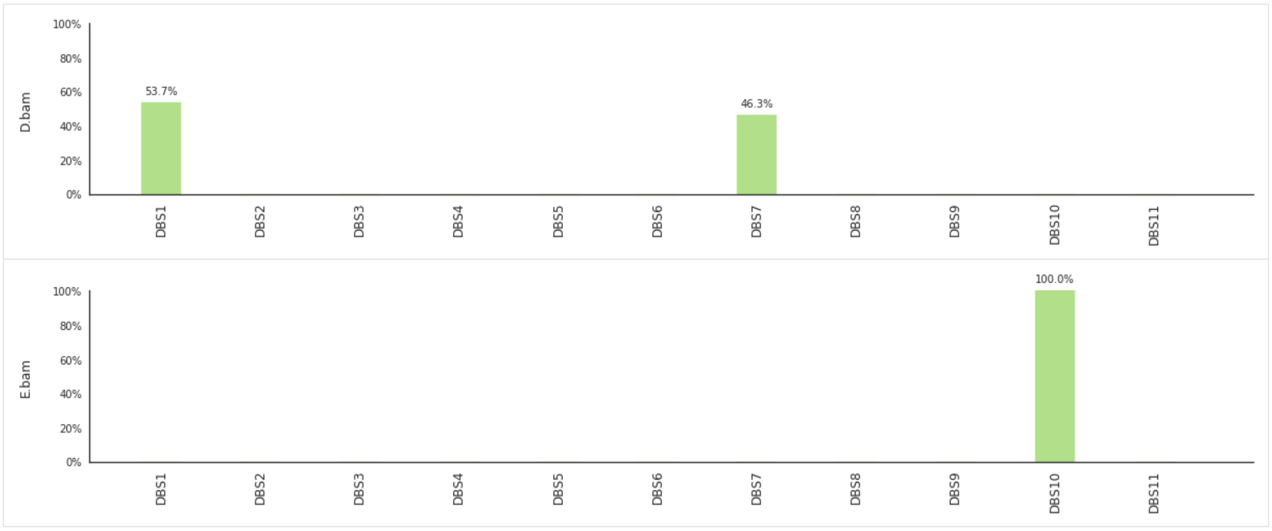
```
[94]: f = im2.postprocess.decompose_SNV_spectra(SNVspectrumDict=SNV_spectra)
```



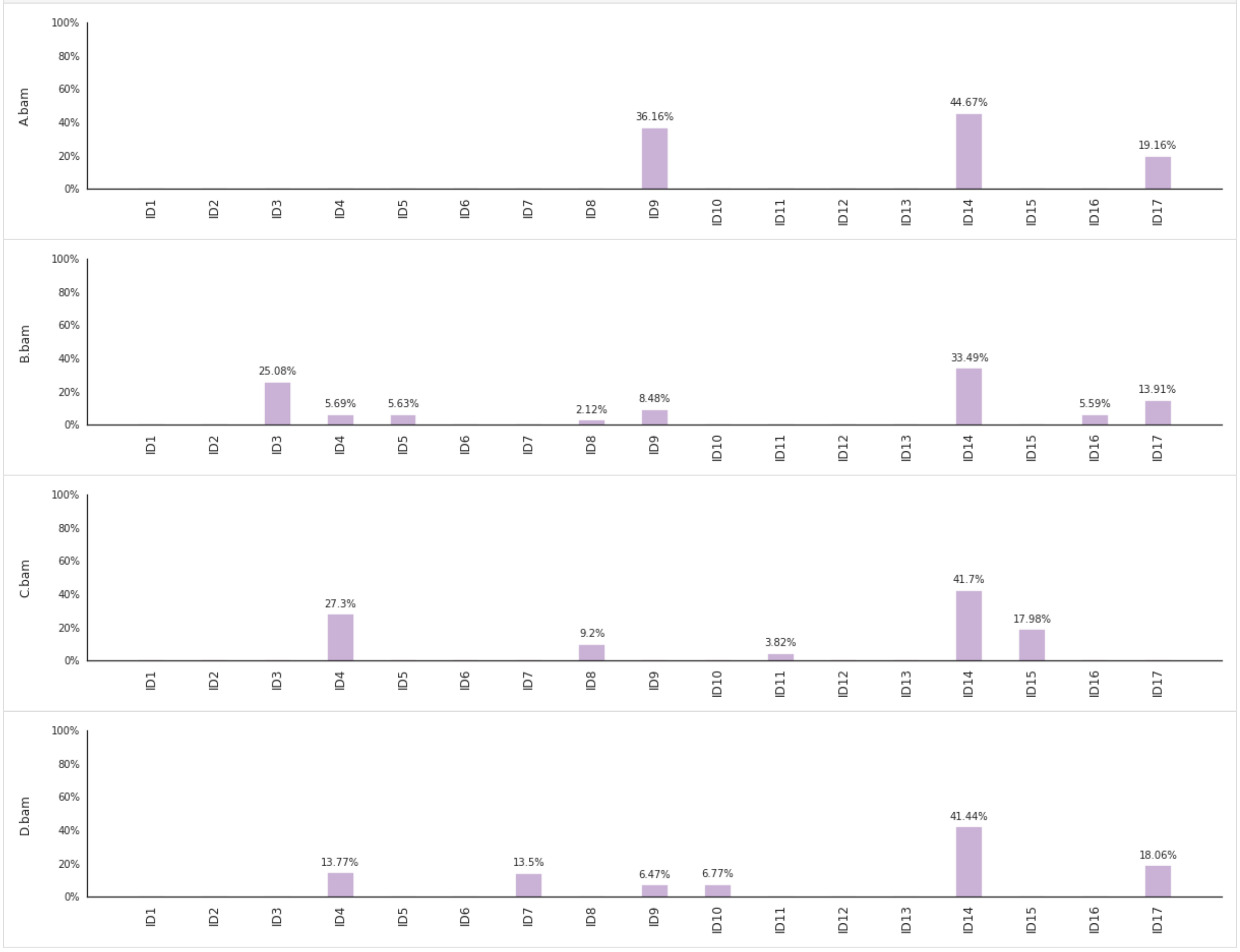


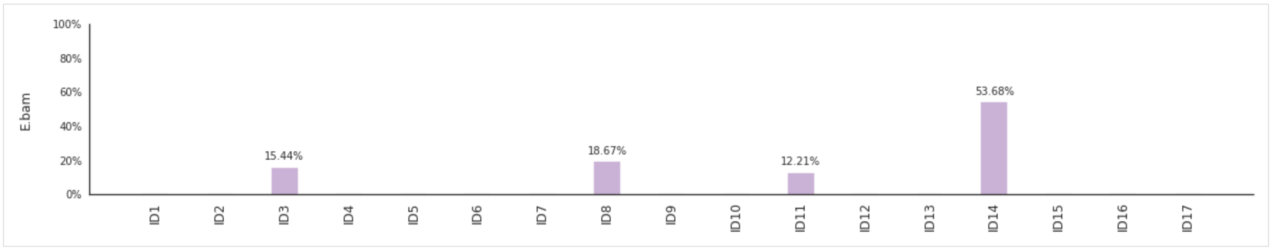
```
[95]: f = im2.postprocess.decompose_DNV_spectra(DNVspectrumDict=DNVspectra)
```





```
[96]: f = im2.postprocess.decompose_indel_spectra(IDspectrumDict=IDspecra)
```





The set of reference signatures used can be modified by setting the argument `signatures_file` in the above functions. The set of mutations included in the decomposition can be set by using the `use_signatures` and `ignore_signatures` arguments.

CHAPTER 7

API reference

7.1 MutationCaller object

class isomut2py.mutationcalling.**MutationCaller**(***kwargs*)

The MutationCaller class is designed to keep all parameter values, directories and filepaths in one place that are needed for the mutation detection and postprocessing of a single or multiple sample(s).

- **List of basic parameters:**
  - ref_fasta: The path to the fasta file of the reference genome. (str)
  - output_dir: The path to a directory that can be used for temporary files and output files. The user must have permission to write the directory. (str)
  - input_dir: The path to the directory, where the bam file(s) of the sample(s) is/are located. (str)
  - bam_filename: A list of the name(s) of the bam file(s) of the sample(s). (Without path, eg. ["sample_1.bam", "sample_2.bam", "sample_3.bam", . . .].) (list of str)
  - samtools_fullpath: The path to samtools on the computer. (default: "samtools") (str)
- **Other parameters with default values:**
  - n_min_block: The approximate number of blocks to partition the analysed genome to for parallel computing. The actual number might be slightly larger that this. (default: 200) (int)
  - n_conc_blocks: The number of blocks to process at the same time. (default: 4) (int)
  - chromosomes: The list of chromosomes to analyse. (default: all chromosomes included included in the reference genome specified in ref_fasta) (list of str)
  - base_quality_limit: The base quality limit used by samtools in order to decide if a base should be included in the pileup file. (default: 30) (int)
  - samtools_flags: The samtools flags to be used for pileup file generation. (default: " -B -d 1000 ") (str)
  - unique_mutations_only: If True, only those mutations are sought that are unique to specific samples. Setting it to False greatly increases computation time, but allows for the detection of

shared mutations and thus the analysis of phylogenetic connections between samples. If True, print_shared_by_all is ignored. (default: True) (boolean)

- print_shared_by_all: If False, mutations that are present in all analysed samples are not printed to the output files. This decreases both memory usage and computation time. (default: False) (boolean)
- min_sample_freq: The minimum frequency of the mutated base at a given position in the mutated sample(s). (default: 0.21) (float)
- min_other_ref_freq: The minimum frequency of the reference base at a given position in the non-mutated sample(s). (default: 0.95) (float)
- cov_limit: The minimum coverage at a given position in the mutated sample(s). (default: 5) (float)
- min_gap_dist_snv: Minimum genomic distance from an identified SNV for a position to be considered as a potential mutation. (default: 0) (int)
- min_gap_dist_indel: Minimum genomic distance from an identified indel for a position to be considered as a potential mutation. (default: 20) (int)
- use_local_realignment: By default, mutation detection is run only once, with the samtools mpileup command with option -B. This turns off the probabilistic realignment of reads while creating the temporary pileup file, which might result in false positives due to alignment error. To filter out these mutations, setting the above parameter to True runs the whole mutation detection pipeline again for possibly mutated positions without the -B option as well, and only those mutations are kept that are still present with the probabilistic realignment turned on. Setting use_local_realignment = True increases runtime. (default: False) (boolean)
- ploidy_info_filepath: Path to the file containing ploidy information (see below for details) of the samples. (default: None) (str)
- constant_ploidy: The default ploidy to be used in regions not contained in ploidy_info_filepath. (default: 2) (int)
- bedfile: Path to a bedfile containing a list of genomic regions, if mutations are only sought in a limited part of the genome. (default: None) (str)
- control_samples: list of the bam filenames of the control samples. Control samples are not expected to have any unique mutations, thus they can be used to optimize the results. (default: None) (array-like)
- FPs_per_genome: the maximum number of false positive mutations allowed in a control sample (default: None) (int)
- mutations: a pandas DataFrame of mutations found in the sample set (default: None) (pandas.DataFrame)
- chrom_length: the length of chromosomes (default: not set) (array-like)
- genome_length: the total length of the genome (default: not set) (int)
- IDspectra: a dictionary containing the indel spectra of all the samples in the dataset (default: not set) (dict with keys=bam_filename and values=array-like)
- DNVspectra: a dictionary containing the dinucleotide variation spectra of all the samples in the dataset (default: not set) (dict with keys=bam_filename and values=array-like)
- SNVspectra: a dictionary containing the single nucleotide variation spectra of all the samples in the dataset (default: not set) (dict with keys=bam_filename and values=array-like)

calculate_DNV_matrix(*unique_only=True*, ***kwargs*)
Calculates the dinucleotide variation spectrum in matrix format from the dataframe of relevant mutations, using the fasta file of the reference genome. Results are stored in the DNVmatrice attribute of the object.

Parameters

- **unique_only** – If True, only unique mutations are plotted for each sample. (default: True) (boolean)
- **kwargs** – keyword arguments for MutationCaller attributes
  - mutations_dataframe: A pandas DataFrame where mutations are listed. (default: not set) (pandas.DataFrame)
  - sample_names: A list of sample names to plot results for, a subset of the list in "bam_filename" attribute. (default: not set) (list)
  - mutations_filaname: the path to the file where mutations are stored (default: not set) (str)

calculate_DNV_spectrum(*unique_only=True*, ***kwargs*)
Calculates the dinucleotide variation spectrum from the dataframe of relevant mutations, using the fasta file of the reference genome. Results are stored in the DNVmatrice attribute of the object.

Parameters

- **unique_only** – If True, only unique mutations are plotted for each sample. (default: True) (boolean)
- **kwargs** – keyword arguments for MutationCaller attributes
  - mutations_dataframe: A pandas DataFrame where mutations are listed. (default: not set) (pandas.DataFrame)
  - sample_names: A list of sample names to plot results for, a subset of the list in "bam_filename" attribute. (default: not set) (list)
  - mutations_filaname: the path to the file where mutations are stored (default: not set) (str)

calculate_SNV_spectrum(*unique_only=True*, ***kwargs*)
Calculates the triplet spectrum from the dataframe of relevant mutations, using the fasta file of the reference genome. Results are stored in the SNVspectra attribute of the object.

Parameters

- **unique_only** – If True, only unique mutations are plotted for each sample. (default: True) (boolean)
- **kwargs** – keyword arguments for MutationCaller attributes
  - mutations_dataframe: A pandas DataFrame where mutations are listed. (default: not set) (pandas.DataFrame)
  - sample_names: A list of sample names to plot results for, a subset of the list in "bam_filename" attribute. (default: not set) (list)
  - mutations_filaname: the path to the file where mutations are stored (default: not set) (str)

calculate_and_plot_DNV_heatmap(*unique_only=True*, *return_string=False*, ***kwargs*)
Calculates and plots the DNV spectra as a heatmap of the samples listed in attribute bam_filename.

Parameters

- **unique_only** – If True, only unique mutations are plotted for each sample. (default: True) (boolean)
- **return_string** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)
- **kwargs** – keyword arguments for MutationCaller attributes

Returns If the return_string value is True, a list of base64 encoded strings of the images. Otherwise, a list of matplotlib figures.

calculate_and_plot_DNV_spectrum(*unique_only=True*, *return_string=False*, *normalize_to_1=False*, ***kwargs*)

Calculates and plots the DNV spectra of the samples listed in attribute bam_filename.

Parameters

- **unique_only** – If True, only unique mutations are plotted for each sample. (default: True) (boolean)
- **normalize_to_1** – If True, results are plotted as percentages, instead of counts. (default: False) (bool)
- **return_string** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)
- **kwargs** – keyword arguments for MutationCaller attributes

Returns If the return_string value is True, a list of base64 encoded strings of the images. Otherwise, a list of matplotlib figures.

calculate_and_plot_SNV_spectrum(*unique_only=True*, *normalize_to_1=False*, *return_string=False*, ***kwargs*)

Calculates and plots the SNV spectra of the samples listed in attribute bam_filename.

Parameters

- **unique_only** – If True, only unique mutations are plotted for each sample. (default: True) (boolean)
- **normalize_to_1** – If True, results are plotted as percentages, instead of counts. (default: False) (bool)
- **return_string** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)
- **kwargs** – keyword arguments for MutationCaller attributes

Returns If the return_string value is True, a list of base64 encoded strings of the images. Otherwise, a list of matplotlib figures.

calculate_and_plot_indel_spectrum(*unique_only=True*, *normalize_to_1=False*, *return_string=False*, ***kwargs*)

Calculates and plots the indel spectra of the samples listed in attribute bam_filename.

Parameters

- **unique_only** – If True, only unique mutations are plotted for each sample. (default: True) (boolean)
- **normalize_to_1** – If True, results are plotted as percentages, instead of counts. (default: False) (bool)
- **return_string** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)

- **kwargs** – keyword arguments for MutationCaller attributes

Returns If the return_string value is True, a list of base64 encoded strings of the images. Otherwise, a list of matplotlib figures.

calculate_indel_spectrum(*unique_only=True*, ***kwargs*)
Calculates the indel spectrum from the dataframe of relevant mutations, using the fasta file of the reference genome. Results are stored in the IDspectra attribute of the object.

Parameters

- **unique_only** – If True, only unique mutations are plotted for each sample. (default: True) (boolean)
- **kwargs** – keyword arguments for MutationCaller attributes

check_pileup(*chrom_list*, *from_pos_list*, *to_pos_list*, *print_original=True*, *savetofile=None*, ***kwargs*)
Loads pileup information for a list of genomic regions.

Parameters

- **chrom_list** – List of chromosomes for the regions. (list of str)
- **from_pos_list** – List of starting positions for the regions. (list of int)
- **to_pos_list** – List of ending positions for the regions. (list of int)
- **print_original** – If True, prints the original string generated with samtools mpileup as well (default: True) (bool)
- **savetofile** – If savetofile is specified, results will be writted to that path. Otherwise, results are written to [output_dir]/checkPileup_tmp.csv (default: None) (str)

Returns df: The processed results of the pileup, containing information on all samples in a pandas.DataFrame.

decompose_DNV_spectra(*sample_names=None*, *unique_only=True*, *signatures_file=None*, *equal_initial_proportions=False*, *tol=0.0001*, *max_iter=1000*, *filter_percent=0*, *filter_count=0*, *keep_top_n=None*, *use_signatures=None*, *ignore_signatures=None*, ***kwargs*)
Run the whole pipeline of decomposing DNV spectra for the samples specified in sample_names.

Parameters

- **sample_names** – The list of sample names analysed. (list of str)
- **unique_only** – If True, only unique mutations are used to construct the original spectrum.
- **signatures_file** – The path to the csv file containing the signature matrix. (str)
- **equal_initial_proportions** – If True, initial weights are initialized to be equal for all reference signature. Otherwise, they are initialized so the signatures with large cosine similarity with the original spectrum get larger weights. (default: False) (bool)
- **tol** – The maximal difference between the proportions for convergence to be True. (default: 0.0001) (float)
- **max_iter** – The maximum number of iterations (default: 1000) (int)
- **filter_percent** – Filter signatures, that contribute less than filter_percent of the mutations in the sample. (default: 0) (float)
- **filter_count** – Filter signatures, that contribute less than filter_count number of mutations in the sample. (default: 0) (int)

- **keep_top_n** – Only keep those signatures that contribute to the mixture with the top keep_top_n number of mutation. (default: None) (int)
- **use_signatures** – Use a specific subset of all signatures. A list of signature names. (default: None) (list of str)
- **ignore_signatures** – Exclude a specific subset of all signatures from the analysis. A list of signature names. (default: None) (list of str)
- **kwargs** – keyword arguments for MutationDetection object attributes

Returns The final proportions of all signatures in the mixture. (Filtered out or not used signatures appear with a proportion of zero.) (numpy.array)

decompose_SNV_spectra(*sample_names=None*, *unique_only=True*, *signatures_file=None*, *equal_initial_proportions=False*, *tol=0.0001*, *max_iter=1000*, *filter_percent=0*, *filter_count=0*, *keep_top_n=None*, *use_signatures=None*, *ignore_signatures=None*, ***kwargs*)

Run the whole pipeline of decomposing SNV spectra for the samples specified in sample_names.

Parameters

- **sample_names** – The list of sample names analysed. (list of str)
- **unique_only** – If True, only unique mutations are used to construct the original spectrum.
- **signatures_file** – The path to the csv file containing the signature matrix. (str)
- **equal_initial_proportions** – If True, initial weights are initialized to be equal for all reference signature. Otherwise, they are initialized so the signatures with large cosine similarity with the original spectrum get larger weights. (default: False) (bool)
- **tol** – The maximal difference between the proportions for convergence to be True. (default: 0.0001) (float)
- **max_iter** – The maximum number of iterations (default: 1000) (int)
- **filter_percent** – Filter signatures, that contribute less than filter_percent of the mutations in the sample. (default: 0) (float)
- **filter_count** – Filter signatures, that contribute less than filter_count number of mutations in the sample. (default: 0) (int)
- **keep_top_n** – Only keep those signatures that contribute to the mixture with the top keep_top_n number of mutation. (default: None) (int)
- **use_signatures** – Use a specific subset of all signatures. A list of signature names. (default: None) (list of str)
- **ignore_signatures** – Exclude a specific subset of all signatures from the analysis. A list of signature names. (default: None) (list of str)
- **kwargs** – keyword arguments for MutationDetection object attributes

Returns The final proportions of all signatures in the mixture. (Filtered out or not used signatures appear with a proportion of zero.) (numpy.array)

decompose_indel_spectra(*sample_names=None*, *unique_only=True*, *signatures_file=None*, *equal_initial_proportions=False*, *tol=0.0001*, *max_iter=1000*, *filter_percent=0*, *filter_count=0*, *keep_top_n=None*, *use_signatures=None*, *ignore_signatures=None*, ***kwargs*)

Run the whole pipeline of decomposing indel spectra for the samples specified in sample_names.

Parameters

- **`sample_names`** – The list of sample names analysed. (list of str)
- **`unique_only`** – If True, only unique mutations are used to construct the original spectrum.
- **`signatures_file`** – The path to the csv file containing the signature matrix. (str)
- **`equal_initial_proportions`** – If True, initial weights are initialized to be equal for all reference signature. Otherwise, they are initialized so the signatures with large cosine similarity with the original spectrum get larger weights. (default: False) (bool)
- **`tol`** – The maximal difference between the proportions for convergence to be True. (default: 0.0001) (float)
- **`max_iter`** – The maximum number of iterations (default: 1000) (int)
- **`filter_percent`** – Filter signatures, that contribute less than filter_percent of the mutations in the sample. (default: 0) (float)
- **`filter_count`** – Filter signatures, that contribute less than filter_count number of mutations in the sample. (default: 0) (int)
- **`keep_top_n`** – Only keep those signatures that contribute to the mixture with the top keep_top_n number of mutation. (default: None) (int)
- **`use_signatures`** – Use a specific subset of all signatures. A list of signature names. (default: None) (list of str)
- **`ignore_signatures`** – Exclude a specific subset of all signatures from the analysis. A list of signature names. (default: None) (list of str)
- **`kwargs`** – keyword arguments for MutationDetection object attributes

Returns The final proportions of all signatures in the mixture. (Filtered out or not used signatures appear with a proportion of zero.) (numpy.array)

`get_details_for_mutations`(*mutations_filename=None*, ***kwargs*)
Get detailed results for the list of mutations contained in the mutations attribute of the object.

Parameters

- **`mutations_filename`** – The path(s) to the file(s) where mutations are stored. (default: None) (list of str)
- **`kwargs`** – keyword arguments for MutationDetection object attributes
  - mutations_dataframe: A pandas DataFrame where mutations are listed. (default: not set) (pandas.DataFrame)

Returns df_joined: A dataframe containing the detailed results. (pandas.DataFrame)

`load_mutations`(*filename=None*)
Loads mutations from a file or a list of files into the mutations attribute.

Parameters **`filename`** – The path to the file, where mutations are stored. A list of paths can be also supplied, in this case, all of them will be loaded to a single dataframe. The mutations attribute of the MutationDetection object will be set to the loaded dataframe. If None, the file [output_dir]/filtered_mutations.csv will be loaded. (default: None) (str)

`optimize_results`(*control_samples*, *FPs_per_genome*, ***kwargs*)
Optimizes the list of detected mutations according to the list of control samples and desired level of false positives set by the user. Filtered results will be loaded to the mutations attribute of the MutationDetection object as a pandas.DataFrame. Optimized values for the score are stored in the attribute optimized_score_values.

Parameters

- **`control_samples`** – List of sample names that should be used as control samples in the sense, that no unique mutations are expected in them. (The sample names listed here must match a subset of the sample names listed in the attribute bam_filename.) (list of str)
- **`FPs_per_genome`** – The total number of false positives tolerated in a control sample. (int)
- **`kwargs`** – possible keyword arguments besides MutationCaller attributes:
  - plot_roc_curve: If True, ROC curves will be plotted as a visual representation of the optimization process. (default: False) (boolean)
  - plot_tuning_curve: If True, tuning curves displaying the number of mutations found in different samples with different score filters will be plotted as a visual representation of the optimization process. (default: False) (boolean)

`plot_DNV_heatmap`(*return_string=False*, ***kwargs*)
Plots the DNV spectra as a heatmap for the samples in attribute bam_filename.

Parameters **`return_string`** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)

Returns If the return_string value is True, a list of base64 encoded strings of the images. Otherwise, a list of matplotlib figures.

`plot_DNV_spectrum`(*return_string=False*, *normalize_to_1=False*, ***kwargs*)
Plots the DNV spectra for the samples in attribute bam_filename.

Parameters

- **`normalize_to_1`** – If True, results are plotted as percentages, instead of counts. (default: False) (bool)
- **`return_string`** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)

Returns If the return_string value is True, a list of base64 encoded strings of the images. Otherwise, a list of matplotlib figures.

`plot_SNV_spectrum`(*return_string=False*, *normalize_to_1=False*, ***kwargs*)
Plots the triplet spectra for the samples in attribute bam_filename.

Parameters

- **`normalize_to_1`** – If True, results are plotted as percentages, instead of counts. (default: False) (bool)
- **`return_string`** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)

Returns If the return_string value is True, a list of base64 encoded strings of the images. Otherwise, a list of matplotlib figures.

`plot_hierarchical_clustering`(*mutations_dataframe=None*, *return_string=False*, *mutations_filename=None*, ***kwargs*)
Generates a heatmap based on the number of shared mutations found in all possible sample pairs. A dendrogram is also added that is the result of hierarchical clustering of the samples.

Parameters

- **`return_string`** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)

- **mutations_filename** – The path to the file, where mutations are stored, if the mutations attribute of the object does not exist, its value will be set to the file defined here. (default: None) (str)
- **mutations_dataframe** – If the mutations are not to be loaded from a file, but are contained in a pandas.DataFrame, this can be supplied with setting the mutations_dataframe parameter. (default: None) (pandas.DataFrame)
- **kwargs** – keyword arguments for MutationDetection object attributes

Returns If the return_string value is True, a base64 encoded string of the image. Otherwise, a matplotlib figure.

plot_indel_spectrum(*return_string=False*, *normalize_to_1=False*, ***kwargs*)
Plots the indel spectra for the samples in attribute bam_filename.

Parameters

- **normalize_to_1** – If True, results are plotted as percentages, instead of counts. (default: False) (bool)
- **return_string** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)

Returns If the return_string value is True, a list of base64 encoded strings of the images. Otherwise, a list of matplotlib figures.

plot_mutation_counts(*unique_only=False*, *return_string=False*, *mutations_filename=None*, *mutations_dataframe=None*, ***kwargs*)
Plots the number of mutations found in all the samples in different ploidy regions.

Parameters

- **unique_only** – If True, only unique mutations are plotted for each sample. (default: False) (boolean)
- **return_string** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)
- **mutations_filename** – The path to the file, where mutations are stored, if the mutations attribute of the object does not exist, its value will be set to the file defined here. (default: None) (str)
- **mutations_dataframe** – If the mutations are not to be loaded from a file, but are contained in a pandas.DataFrame, this can be supplied with setting the mutations_dataframe parameter. (default: None) (pandas.DataFrame)
- **kwargs** – possible keyword arguments besides MutationCaller attributes
  - control_samples: List of sample names that should be used as control samples in the sense, that no unique mutations are expected in them. (The sample names listed here must match a subset of the sample names listed in bam_filename.) (list of str)

Returns If the return_string value is True, a base64 encoded string of the image. Otherwise, a matplotlib figure.

plot_rainfall(*unique_only=True*, *return_string=False*, ***kwargs*)
Plots the rainfall plot of mutations found in the samples listed in the attribute bam_filename. Displaying rainfall plots is a good practice to detect mutational clusters throughout the genome. The horizontal axis is the genomic position of each mutation, while the vertical shows the genomic distance between the mutation and the previous one. Thus mutations that are clustered together appear close to each other horizontally and on the lower part of the plot vertically.

Parameters

- **`unique_only`** – If True, only unique mutations are plotted for each sample. (default: True) (boolean)
- **`return_string`** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)
- **`kwargs`** – keyword arguments for MutationCaller attributes
  - mutations_dataframe: A pandas DataFrame where mutations are listed. (default: not set) (pandas.DataFrame)
  - sample_names: A list of sample names to plot results for, a subset of the list in "bam_filename" attribute. (default: not set) (list)
  - mut_types: list of mutation types to display (default: not set, displaying SNVs, insertions and deletions) (a list containing any combination of these items: ['SNV', 'INS', 'DEL'])
  - plot_range: The genomic range to plot. (default: not set) (str, example: "chr9:2342-24124")

Returns If the return_string value is True, a list of base64 encoded strings of the images. Otherwise, a list of matplotlib figures.

`run_isomut2_mutdet`(***kwargs*)

Runs IsoMut2 mutation detection pipeline on the MutationDetection object, using parameter values specified in the respective attributes of the object.

Parameters **`kwargs`** – keyword arguments for MutationDetection object attributes

7.2 PloidyEstimator object

class `isomut2py.ploidyestimation.`**`PloidyEstimator`**(***kwargs*)

The PloidyEstimator class is designed to keep all parameter values, directories and filepaths in one place that are needed for the ploidy analysis of a single sample.

- **List of basic parameters:**
  - ref_fasta: The path to the fasta file of the reference genome. (str)
  - output_dir: The path to a directory that can be used for temporary files and output files. The user must have permission to write the directory. (str)
  - input_dir: The path to the directory, where the bam file(s) of the sample(s) is/are located. (str)
  - bam_filename: The name of the bam file of the sample. (Without path, eg. "sample_1.bam".) (str)
  - samtools_fullpath: The path to samtools on the computer. (default: "samtools") (str)
- **Other parameters with default values:**
  - n_min_block: The approximate number of blocks to partition the analysed genome to for parallel computing. The actual number might be slightly larger that this. (default: 200) (int)
  - n_conc_blocks: The number of blocks to process at the same time. (default: 4) (int)
  - chromosomes: The list of chromosomes to analyse. (default: all chromosomes included included in the reference genome specified in ref_fasta) (list of str)
  - windowsize: The windowsize used for initial coverage smoothing of the bam file with a moving average method. Setting it too large might disguise CNV effects. (default: 10000) (int)

- shiftsize: The shiftsize used for the moving average method of the initial coverage smoothing procedure. MUST be smaller than windowsize. (default: 3000) (int)
- min_noise: The minimum frequency of non-reference or reference bases detected for a position to be considered for LOH detection. Setting it too small will result in poor noise filtering, setting it too large will result in a decreased number of measurement points. (default: 0.1) (float in range(0,1))
- base_quality_limit: The base quality limit used by samtools in order to decide if a base should be included in the pileup file. (default: 0) (int)
- print_every_nth: Even though LOH detection is limited to the positions with a noise level larger that min_noise, ploidy estimation is based on all the genomic positions meeting the above set criteria. By setting the attribute print_every_nth, the number of positions used can be controlled. Setting it large will result in overlooking ploidy variations in shorter genomic ranges, while setting it too small can cause an increase in both memory usage and computation time. Decrease only if a relatively short genome is analysed. (default: 100) (int)
- windowsize_PE: The windowsize used for actual ploidy estimation after the initial coverage smoothing. (default: 1000000) (int)
- shiftsize_PE: The shiftsize used for actual ploidy estimation after the initial coverage smoothing. MUST be smaller than windowsize_PE. (default: 50000) (int)
- cov_max: The maximum coverage in a genomic position that can be considered for ploidy estimation. Alignment errors might cause certain genomic positions to have an enormous coverage. These outliers are ignored, when cov_max is set. The value must be set in agreement with the average sequencing depth. Using a low value for a deeply sequenced sample can result in a decreased number of positions to be analysed. (default: 200) (int)
- cov_min: The minimum coverage in a genomic position that can be considered for ploidy estimation. Sequencing noise might cause certain genomic positions to have very low coverage, frequently merely from misaligned reads. These outliers are ignored, when cov_min is set. The value must be set in agreement with the average sequencing depth. Using a high value for a shallowly sequenced sample can result in a decreased number of positions to be analysed. (default: 5) (int)
- hc_percentile: The haploid coverage of the sequenced sample is estimated multiple times by fitting a mixture model to the coverage distribution of the sample. The actual value is chosen as a statistical measure of these multiple results, set by the value of hc_percentile. For example, setting hc_percentile to 50 results in using the median of the results. For more details on the suggested values, see Pipek et al. 2018. (default: 75) (int)
- compare_to_bed: The path to a bed file to compare ploidy estimation results to. (default: None) (str)
- samtools_flags: The samtools flags to be used for pileup file generation. (default: ” -B -d 1000 “) (str)
- user_defined_hapcov: During ploidy estimation, the haploid coverage is estimated from the coverage distribution of the sample. In some cases, the estimation might not find the real value of the haploid coverage. In these situations, supplying an estimate of the haploid coverage manually might improve the overall ploidy estimation results. If you have a generally diploid genome, using the half of the average coverage can be a good starting point. If user_defined_hapcov is set, hc_percentile is ignored. (default: None) (float)

PE_on_chrom(*chrom*, ***kwargs*)

Runs the whole ploidy estimation pipeline on a given chromosome, using the appropriate attributes of the PloidyEstimation object by running PE_on_range() multiple times. Prints the results to the file: [self.output_dir]/PE_fullchrom_[chrom].txt.

Parameters

- **`chrom`** – the name of the chromosome to analyse (str)
- **`kwargs`** – keyword arguments for PloidyEstimator object

`PE_on_whole_genome`(***kwargs*)

Runs the whole ploidy estimation pipeline on the whole genome by running PE_on_chrom() on all chromosomes.

Parameters **`kwargs`** – keyword arguments for PloidyEstimator object

`compare_with_other`(*other*, *minLen=2000*, *minQual=0.1*)

Compare ploidy estimation results with another PloidyEstimation object or a bed file.

Parameters

- **`other`** – The other PloidyEstimation object or the path to the other bedfile. (isomut2py.PloidyEstimation or str)
- **`minLen`** – The minimum length of a region to be considered different from the other object or file. (int)
- **`minQual`** – The minimum quality of a region to be considered different from the other object or file. (float)

`estimate_hapcov_infmix`(*level=0*, ***kwargs*)

Estimates the haploid coverage of the sample from the appropriate attributes of the PloidyEstimation object. If the user_defined_hapcov attribute is set manually, it sets the value of estimated_hapcov to that. Otherwise, a many-component (20) Gaussian mixture model is fitted to the coverage histogram of the sample 10 times. Each time, the haploid coverage is estimated from the center of the component with the maximal weight in the model. The final estimate of the haploid coverage is calculated as the qth percentile of the 10 measurements, with q = hc_percentile. Sets the "estimated_hapcov" attribute to the calculated haploid coverage and the "coverage_sample" attribute to a 2000-element sample of the coverage distribution.

Parameters

- **`level`** – the level of indentation used in verbose output (default: 0) (int)
- **`kwargs`** – keyword arguments for PloidyEstimator object

`fit_gaussians`(*level=0*, ***kwargs*)

Fits a 7-component Gaussian mixture model to the coverage distribution of the sample, using the appropriate attributes of the PloidyEstimation object. The center of the first Gaussian is initialized from a narrow region around the value of the estimated_hapcov attribute. The centers of the other Gaussians are initialized in a region around the value of estimated_hapcov multiplied by consecutive whole numbers.

The parameters of the fitted model (center, sigma and weight) for all seven Gaussians are both saved to the GaussDistParams.pkl file (in output_dir, for later reuse) and set as the value of the distribution_dict attribute.

Parameters

- **`level`** – the level of indentation used in verbose output (default: 0) (int)
- **`kwargs`** – keyword arguments for PloidyEstimator object

`generate_HTML_report_for_ploidy_est`(***kwargs*)

Generates a HTML file with figures displaying the results of ploidy estimation and saves it to output_dir/PEreport.html.

Parameters **`kwargs`** – keyword arguments for PloidyEstimator object

get_bed_format_for_sample(***kwargs*)
Creates bed file of constant ploidies for a given sample from a file of positional ploidy data. If the ownbed_filepath attribute of the PloidyEstimation object is set, saves the bedfile to the path specified there. Otherwise, saves it to the output_dir with the "_ploidy.bed" suffix. Also sets the bed_dataframe attribute to the pandas.Dataframe containing the bed file.

Parameters **kwargs** – keyword arguments for PloidyEstimator object

get_coverage_distribution(***kwargs*)
Sets the coverage_sample attribute of the PloidyEstimation object to the coverage distribution obtained from the temporary files created by __PE_prepare_temp_files(). Positions are filtered according to the attributes of the PloidyEstimation object. The number of positions in the final sample is decreased to 2000 for faster inference.

Parameters **kwargs** – keyword arguments for PloidyEstimator object

Returns A 2000-element sample of the coverage distribution.

load_bedfile_from_file(*filename=None*, ***kwargs*)
Loads the bedfile containing previous ploidy estimated for the given sample from the path specified in filename. The dataframe will be stored in the "bed_dataframe" attribute.

Parameters

- **filename** – The path to the bedfile. (default: [output_dir]/[bam_filename]_ploidy.bed) (str)
- **kwargs** – keyword arguments for PloidyEstimator object

load_cov_distribution_parameters_from_file(*filename=None*, ***kwargs*)
Loads the parameters of the seven fitted Gaussians to the coverage distribution of the sample from the specified filename (that was saved with pickle beforehand). If one such file is available, the computationally expensive ploidy estimation process can be skipped. The parameter values will be stored in the attribute "distribution_dict" as a dictionary.

Parameters

- **filename** – The path to the file with the coverage distribution parameters. (default: [output_dir]/GaussDistParams.pkl) (str)
- **kwargs** – keyword arguments for PloidyEstimator object

plot_coverage_distribution(***kwargs*)
Plot the coverage distribution of the sample.

Parameters **kwargs** – keyword arguments for PloidyEstimator object

Returns a matplotlib figure of the coverage distribution

plot_karyotype_for_all_chroms(*return_string=False*, ***kwargs*)
Plots karyotype information (coverage, estimated ploidy, estimated LOH, reference base frequencies) about the sample for all analysed chromosomes.

Parameters

- **return_string** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: True) (bool)
- **kwargs** – keyword arguments for PloidyEstimator object

Returns If the return_string value is True, a list of base64 encoded strings of the images. Otherwise, a list of matplotlib figures.

plot_karyotype_summary(***kwargs*)
Plots a simple karyotype summary for the whole genome. (Details coming soon.)

Parameters **kwargs** – keyword arguments for PloidyEstimator object

Returns a matplotlib figure of the plot

run_ploidy_estimation(***kwargs*)
Runs the whole ploidy estimation pipeline on the PloidyEstimation object.

Parameters **kwargs** – keyword arguments for PloidyEstimator object

7.3 Functions for plotting

Most of these functions can be called as methods of a *MutationCaller object*, but to make it more straightforward to analyse any set of (external) mutations, they can also be called individually.

isomut2py.plot.**generate_HTML_report_for_ploidy_est**(*chromosomes*, *output_dir*, *min_noise=nan*)
Generates a HTML file with figures displaying the results of ploidy estimation and saves it to output_dir/PEreport.html.

Parameters

- **chromosomes** – list of chromosomes in the genome (list of str)
- **output_dir** – the path to the directory where PE_fullchrom_[chrom].txt files are located (str)
- **min_noise** – the minimal B-allele frequency for a position to be included in the analyses (default: numpy.nan) (float)

isomut2py.plot.**plot_DNV_heatmap**(*matrixDict*, *return_string=False*, *normalize_to_1=False*)
Plot DNVs (dinucleotide variations) as a heatmap for a database of mutations.

Parameters

- **matrixDict** – a dictionary containing 12x12 element matrices as values and sample names as keys (dictionary)
- **return_string** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)
- **normalize_to_1** – If True, results are plotted as percentages, instead of counts. (default: False) (bool)

Returns If the return_string value is True, a base64 encoded string of the image. Otherwise, nothing.

isomut2py.plot.**plot_DNV_spectrum**(*spectrumDict*, *return_string=False*, *normalize_to_1=False*)
Plots the DNV spectrum, given a dictionary containing the spectra as values.

Parameters

- **spectrumDict** – a dictionary containing DNV spectra as values and sample names as keys (dictionary)
- **normalize_to_1** – If True, results are plotted as percentages, instead of counts. (default: False) (bool)
- **return_string** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)

Returns If the return_string value is True, a base64 encoded string of the image. Otherwise, a list of matplotlib figures.

isomut2py.plot.**plot_SNV_spectrum**(*spectrumDict*, *return_string=False*, *normalize_to_1=False*)

Plots the triplet spectrum for a list of 96-element vectors defined in spectrumDict.

Parameters

- **spectrumDict** – a dictionary containing spectra as values and sample names as keys (dictionary)
- **normalize_to_1** – If True, results are plotted as percentages, instead of counts. (default: False) (bool)
- **return_string** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)

Returns If the return_string value is True, a base64 encoded string of the image. Otherwise, a list of matplotlib figures.

isomut2py.plot.**plot_coverage_distribution**(*cov_sample=None*, *chromosomes=None*, *output_dir=None*, *cov_max=None*, *cov_min=None*, *distribution_dict=None*)

Plot the coverage distribution of the sample.

Parameters

- **cov_sample** – a sample of the coverage distribution (default: None) (array-like)
- **chromosomes** – the list of chromosomes in the genome (default: None) (list of str)
- **output_dir** – the path to the directory where PE_fullchrom_[chrom].txt files are located (default: None) (str)
- **cov_max** – the maximum value for the coverage for a position to be included on the plot (default: None) (int)
- **cov_min** – the minimum value for the coverage for a position to be included on the plot (default: None) (int)
- **distribution_dict** – a dictionary containing the fitted parameters of the coverage distribution (default: None) (dictionary with keys: 'mu', 'sigma', 'p')

isomut2py.plot.**plot_hierarchical_clustering**(*sample_names=None*, *mutations_dataframe=None*, *mutations_filename=None*, *output_dir=None*, *return_string=False*, *method='average'*)

Generates a heatmap based on the number of shared mutations found in all possible sample pairs. A dendrogram is also added that is the result of hierarchical clustering of the samples.

Parameters

- **mutations_dataframe** – The dataframe containing the mutations. (default: None) (pandas.DataFrame)
- **sample_names** – list of samples names to plot mutation counts for (default: None) (list of str)
- **return_string** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)
- **mutations_filename** – The path to the file, where mutations are stored, if the mutations attribute of the object does not exist, its value will be set to the file defined here. (default: None) (str)

- **`output_dir`** – the path to the directory where mutation tables are located (default: None) (str)
- **`method`** – method used for seaborn hierarchical clustering (default: 'average') ("single", "complete", "average", "weighted", "median", "ward")

Returns If the return_string value is True, a base64 encoded string of the image. Otherwise, a list of matplotlib figures.

`isomut2py.plot.`**`plot_indel_spectrum`**(*spectrumDict*, *return_string=False*, *normalize_to_1=False*)

Plots the indel spectrum, given a dictionary containing 83-element vectors as values.

Parameters

- **`spectrumDict`** – a dictionary containing spectra as values and sample names as keys (dictionary)
- **`normalize_to_1`** – If True, results are plotted as percentages, instead of counts. (default: False) (bool)
- **`return_string`** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: True) (bool)

Returns If the return_string value is True, a base64 encoded string of the image. Otherwise, a list of matplotlib figures.

`isomut2py.plot.`**`plot_karyotype_for_all_chroms`**(*chromosomes*, *output_dir*, *return_string=False*)

Plots karyotype information (coverage, estimated ploidy, estimated LOH, reference base frequencies) about the sample for all analysed chromosomes.

Parameters

- **`chromosomes`** – the list of chromosomes to plot (list of str)
- **`output_dir`** – the path to the directory where PE_fullchrom_[chrom].txt files are stored. (str)
- **`return_string`** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: True) (bool)

Returns If the return_string value is True, a base64 encoded string of the image. Otherwise, nothing.

`isomut2py.plot.`**`plot_karyotype_for_chrom`**(*chrom*, *df*, *return_string=True*)

Plots karyotype information (coverage, estimated ploidy, estimated LOH, reference base frequencies) about the sample for a given chromosome.

Parameters

- **`chrom`** – The chromosome to plot. (str)
- **`df`** – The dataframe containing ploidy and LOH information. (pandas.DataFrame)
- **`return_string`** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: True) (bool)

Returns If the return_string value is True, a base64 encoded string of the image. Otherwise, a matplotlib figure.

`isomut2py.plot.`**`plot_karyotype_summary`**(*haploid_coverage*, *chromosomes*, *chrom_length*, *output_dir*, *bed_filename*, *bed_file_sep=', '*, *binsize=1000000*, *overlap=50000*, *cov_min=5*, *cov_max=200*, *min_PL_length=3000000*, *chroms_with_text=None*)

Plots karyotype summary for the whole genome with data preparation.

Parameters

- **`haploid_coverage`** – the average coverage of haploid regions (or the half of that of diploid regions)
- **`chromosomes`** – list of chromosomes in the genome (list of str)
- **`chrom_length`** – list of chromosome lengths (list of int)
- **`output_dir`** – the path to the directory where PE_fullchrom_[chrom].txt files are located (str)
- **`bed_filename`** – the path to the bed file of the sample with ploidy and LOH information (str)
- **`bed_file_sep`** – bed file separator (default: ‘,’) (str)
- **`binsize`** – the binsize used for moving average (default: 1000000) (int)
- **`overlap`** – the overlap used for moving average (default: 50000) (int, smaller than binsize)
- **`cov_min`** – the minimum coverage for a position to be included (default: 5) (int)
- **`cov_max`** – the maximum coverage for a position to be included (default: 2000) (int)
- **`min_PL_length`** – the minimal length of a region to be plotted (default: 3000000) (int)
- **`chroms_with_text`** – the list of chromosomes to be indicated with text on the plot (list of str) (If there are many short chromosomes or they have long names, it is useful to only indicate a few with text on the plot.)

Returns a matplotlib figure

`isomut2py.plot.`**`plot_mutation_counts`**(*sample_names=None*, *mutations_dataframe=None*, *unique_only=False*, *return_string=False*, *mutations_filename=None*, *output_dir=None*, *control_samples=None*)

Plots the number of mutations found in all the samples in different ploidy regions.

Parameters

- **`mutations_dataframe`** – The dataframe containing the mutations. (default: None) (pandas.DataFrame)
- **`sample_names`** – list of samples names to plot mutation counts for (default: None) (list of str)
- **`unique_only`** – If True, only unique mutations are plotted for each sample. (default: False) (boolean)
- **`return_string`** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)
- **`mutations_filename`** – The path to the file, where mutations are stored, if the mutations attribute of the object does not exist, its value will be set to the file defined here. (default: None) (str)
- **`output_dir`** – the path to the directory where mutation tables are located (default: None) (str)
- **`control_samples`** – List of sample names that should be used as control samples in the sense, that no unique mutations are expected in them. (The sample names listed here must match a subset of the sample names listed in bam_filename.) (list of str)

Returns If the return_string value is True, a base64 encoded string of the image. Otherwise, a list of matplotlib figures.

`isomut2py.plot.`**`plot_rainfall`**(*mutations_dataframe, chromosomes=None, chrom_length=None, ref_fasta=None, sample_names=None, return_string=False, muttypes=['SNV', 'INS', 'DEL'], unique_only=True, plot_range=None*)

Plot a rainfall plot of the mutations. The horizontal axis is the genomic position of each mutation and the vertical axis is the genomic difference measured from the previous mutation.

Parameters

- **`mutations_dataframe`** – the pandas.DataFrame containing all mutations (pandas.DataFrame)
- **`chromosomes`** – a list of chromosomes to be plotted (default: None) (list of str)
- **`chrom_length`** – a list of chrom lengths (default: None) (list of int)
- **`ref_fasta`** – the path to the reference fasta file (default: None) (str)
- **`sample_names`** – the list of sample names to be plotted (default: None) (str)
- **`return_string`** – If True, only a temporary plot is generated and its base64 code is returned, that can be included in HTML files. (default: False) (bool)
- **`muttypes`** – the list of mutation types to be plotted (default: ["SNV", "INS", "DEL"]) (any elements of the default list)
- **`unique_only`** – If True, only unique mutations are plotted for each sample. (default: False) (boolean)
- **`plot_range`** – the genomic range to be plotted (default: None, the whole genome is plotted) (str, example: "chr9:123134-143441414")

Returns If the return_string value is True, a base64 encoded string of the image. Otherwise, a list of matplotlib figures.

7.4 Functions for postprocessing lists of mutations

`isomut2py.postprocess.`**`calculate_DNV_matrix`**(*mutations_dataframe=None, sample_names=None, unique_only=True, output_dir=None, mutations_filename=None*)

Calculates the DNV matrix from a dataframe of relevant mutations.

Parameters

- **`mutations_dataframe`** – Dataframe containing the list of mutations to be considered. (default: None) (pandas.DataFrame)
- **`sample_names`** – the list of sample names included in the analysis (default: None) (list of str)
- **`unique_only`** – if True, only unique mutations are considered for the spectrum (default: True) (bool)
- **`output_dir`** – path to the directory where mutation tables are located (default: None) (str)
- **`mutations_filename`** – path to the mutation table(s) (default: None) (list of str)

Returns A dictionary containing DNV matrices as values and sample names as keys. (str: numpy.array)

`isomut2py.postprocess.`**`calculate_DNV_spectrum`**(*mutations_dataframe=None*, *sample_names=None*, *unique_only=True*, *output_dir=None*, *mutations_filename=None*)

Calculates the indel spectrum from a dataframe of relevant mutations.

Parameters

- **`mutations_dataframe`** – Dataframe containing the list of mutations to be considered. (default: None) (pandas.DataFrame)
- **`sample_names`** – the list of sample names included in the analysis (default: None) (list of str)
- **`unique_only`** – if True, only unique mutations are considered for the spectrum (default: True) (bool)
- **`output_dir`** – path to the directory where mutation tables are located (default: None) (str)
- **`mutations_filename`** – path to the mutation table(s) (default: None) (list of str)

Returns A dictionary containing DNV spectra arrays (the counts for each mutation type) as values and sample names as keys. (str: numpy.array)

`isomut2py.postprocess.`**`calculate_SNV_spectrum`**(*ref_fasta*, *mutations_dataframe=None*, *sample_names=None*, *unique_only=True*, *chromosomes=None*, *output_dir=None*, *mutations_filename=None*)

Calculates the triplet spectrum from a dataframe of relevant mutations, using the fasta file of the reference genome.

Parameters

- **`ref_fasta`** – the path to the reference genome fasta file (str)
- **`mutations_dataframe`** – Dataframe containing the list of mutations to be considered. (default: None) (pandas.DataFrame)
- **`sample_names`** – the list of sample names included in the analysis (default: None) (list of str)
- **`unique_only`** – if True, only unique mutations are considered for the spectrum (default: True) (bool)
- **`chromosomes`** – list of chromosomes in the analysis (default: None) (list of str)
- **`output_dir`** – path to the directory where mutation tables are located (default: None) (str)
- **`mutations_filename`** – path to the mutation table(s) (default: None) (list of str)

Returns A dictionary containing 96-element vectors (the counts for each mutation type) as values and sample names as keys. (str: numpy.array)

`isomut2py.postprocess.`**`calculate_indel_spectrum`**(*ref_fasta*, *mutations_dataframe=None*, *sample_names=None*, *unique_only=True*, *chromosomes=None*, *output_dir=None*, *mutations_filename=None*)

Calculates the indel spectrum from a dataframe of relevant mutations, using the fasta file of the reference genome.

Parameters

- **ref_fasta** – the path to the reference genome fasta file (str)
- **mutations_dataframe** – Dataframe containing the list of mutations to be considered. (default: None) (pandas.DataFrame)
- **sample_names** – the list of sample names included in the analysis (default: None) (list of str)
- **unique_only** – if True, only unique mutations are considered for the spectrum (default: True) (bool)
- **chromosomes** – list of chromosomes in the analysis (default: None) (list of str)
- **output_dir** – path to the directory where mutation tables are located (default: None) (str)
- **mutations_filename** – path to the mutation table(s) (default: None) (list of str)

Returns A dictionary containing 83-element vectors (the counts for each mutation type) as values and sample names as keys. (str: numpy.array)

`isomut2py.postprocess.`**`check_pileup`**(*chrom_list*, *from_pos_list*, *to_pos_list*, *ref_fasta*, *input_dir*, *bam_filename*, *output_dir=”*, *samtools_fullpath='samtools'*, *base_quality_limit=30*, *samtools_flags=' -B -d 1000 '*, *print_original=True*, *filename=None*)

Loads pileup information for a list of genomic regions.

Parameters

- **chrom_list** – List of chromosomes for the regions. (list of str)
- **from_pos_list** – List of starting positions for the regions. (list of int)
- **to_pos_list** – List of ending positions for the regions. (list of int)
- **ref_fasta** – the path to the reference genome fasta file (str)
- **bam_filename** – the filenames for the bam files of the investigated samples (list of str)
- **output_dir** – path to the directory where temporary files should be saved (default: ‘’) (str)
- **samtools_fullpath** – path to samtools on the computer (default: “samtools’) (str)
- **base_quality_limit** – base quality limit for the pileup generation (default: 30) (int)
- **samtools_flags** – additional flags for samtools (default: ‘ -B -d ‘ + str(SAMTOOLS_MAX_DEPTH) + ‘ ‘) (str)
- **print_original** – If True, prints the original string generated with samtools mpileup as well (default: True) (bool)
- **filename** – If filename is specified, results will be writted to that path. Otherwise, results are written to [output_dir]/checkPileup_tmp.csv (default: None) (str)

Returns df: The processed results of the pileup, containing information on all samples in a pandas.DataFrame.

isomut2py.postprocess.**decompose_DNV_spectra**(*DNVspectrumDict*, *sample_names=None*, *signatures_file=None*, *equal_initial_proportions=False*, *tol=0.0001*, *max_iter=1000*, *filter_percent=0*, *filter_count=0*, *keep_top_n=None*, *use_signatures=None*, *ignore_signatures=None*)

Run the whole pipeline of decomposing DNV spectra for the samples specified in sample_names.

Parameters

- **DNVspectrumDict** – dictionary containing DNV spectra as values and sample names as keys (dictionary)
- **sample_names** – The list of sample names analysed. (list of str)
- **unique_only** – If True, only unique mutations are used to construct the original spectrum.
- **signatures_file** – The path to the csv file containing the signature matrix. (str)
- **equal_initial_proportions** – If True, initial weights are initialized to be equal for all reference signature. Otherwise, they are initialized so the signatures with large cosine similarity with the original spectrum get larger weights. (default: False) (bool)
- **tol** – The maximal difference between the proportions for convergence to be True. (default: 0.0001) (float)
- **max_iter** – The maximum number of iterations (default: 1000) (int)
- **filter_percent** – Filter signatures, that contribute less than filter_percent of the mutations in the sample. (default: 0) (float)
- **filter_count** – Filter signatures, that contribute less than filter_count number of mutations in the sample. (default: 0) (int)
- **keep_top_n** – Only keep those signatures that contribute to the mixture with the top keep_top_n number of mutation. (default: None) (int)
- **use_signatures** – Use a specific subset of all signatures. A list of signature names. (default: None) (list of str)
- **ignore_signatures** – Exclude a specific subset of all signatures from the analysis. A list of signature names. (default: None) (list of str)

Returns The final proportions of all signatures in the mixture. (Filtered out or not used signatures appear with a proportion of zero.) (numpy.array)

isomut2py.postprocess.**decompose_SNV_spectra**(*SNVspectrumDict*, *sample_names=None*, *signatures_file=None*, *equal_initial_proportions=False*, *tol=0.0001*, *max_iter=1000*, *filter_percent=0*, *filter_count=0*, *keep_top_n=None*, *use_signatures=None*, *ignore_signatures=None*)

Run the whole pipeline of decomposing SNV spectra for the samples specified in sample_names.

Parameters

- **SNVspectrumDict** – dictionary containing SNV spectra as values and sample names as keys (dictionary)
- **sample_names** – The list of sample names analysed. (list of str)

- **unique_only** – If True, only unique mutations are used to construct the original spectrum.
- **signatures_file** – The path to the csv file containing the signature matrix. (str)
- **equal_initial_proportions** – If True, initial weights are initialized to be equal for all reference signature. Otherwise, they are initialized so the signatures with large cosine similarity with the original spectrum get larger weights. (default: False) (bool)
- **tol** – The maximal difference between the proportions for convergence to be True. (default: 0.0001) (float)
- **max_iter** – The maximum number of iterations (default: 1000) (int)
- **filter_percent** – Filter signatures, that contribute less than filter_percent of the mutations in the sample. (default: 0) (float)
- **filter_count** – Filter signatures, that contribute less than filter_count number of mutations in the sample. (default: 0) (int)
- **keep_top_n** – Only keep those signatures that contribute to the mixture with the top keep_top_n number of mutation. (default: None) (int)
- **use_signatures** – Use a specific subset of all signatures. A list of signature names. (default: None) (list of str)
- **ignore_signatures** – Exclude a specific subset of all signatures from the analysis. A list of signature names. (default: None) (list of str)

Returns The final proportions of all signatures in the mixture. (Filtered out or not used signatures appear with a proportion of zero.) (numpy.array)

`isomut2py.postprocess.`**`decompose_indel_spectra`**(*IDspectrumDict*, *sample_names=None*, *signatures_file=None*, *equal_initial_proportions=False*, *tol=0.0001*, *max_iter=1000*, *filter_percent=0*, *filter_count=0*, *keep_top_n=None*, *use_signatures=None*, *ignore_signatures=None*)

Run the whole pipeline of decomposing indel spectra for the samples specified in sample_names.

Parameters

- **IDspectrumDict** – dictionary containing indel spectra as values and sample names as keys (dictionary)
- **sample_names** – The list of sample names analysed. (list of str)
- **signatures_file** – The path to the csv file containing the signature matrix. (str)
- **equal_initial_proportions** – If True, initial weights are initialized to be equal for all reference signature. Otherwise, they are initialized so the signatures with large cosine similarity with the original spectrum get larger weights. (default: False) (bool)
- **tol** – The maximal difference between the proportions for convergence to be True. (default: 0.0001) (float)
- **max_iter** – The maximum number of iterations (default: 1000) (int)
- **filter_percent** – Filter signatures, that contribute less than filter_percent of the mutations in the sample. (default: 0) (float)
- **filter_count** – Filter signatures, that contribute less than filter_count number of mutations in the sample. (default: 0) (int)

- **keep_top_n** – Only keep those signatures that contribute to the mixture with the top keep_top_n number of mutation. (default: None) (int)
- **use_signatures** – Use a specific subset of all signatures. A list of signature names. (default: None) (list of str)
- **ignore_signatures** – Exclude a specific subset of all signatures from the analysis. A list of signature names. (default: None) (list of str)

Returns The final proportions of all signatures in the mixture. (Filtered out or not used signatures appear with a proportion of zero.) (numpy.array)

`isomut2py.postprocess.`**`get_details_for_mutations`**(*ref_fasta*, *input_dir*, *bam_filename*, *mutations_dataframe=None*, *output_dir=None*, *mutations_filename=None*, *samtools_fullpath='samtools'*, *base_quality_limit=30*, *samtools_flags=' -B -d 1000 '*)

Get detailed results for the list of mutations contained in the mutations attribute of the object.

Parameters

- **ref_fasta** – the path to the reference genome fasta file (str)
- **input_dir** – the path to the directory where bam files are located (str)
- **bam_filename** – the filenames for the bam files of the investigated samples (list of str)
- **mutations_dataframe** – a pandas.DataFrame where mutations are stored (default: None) (pandas.DataFrame)
- **output_dir** – path to the directory where temporary files should be saved (default: '') (str)
- **mutations_filename** – path to the file(s) where mutations are stored (default: None) (list of str)
- **samtools_fullpath** – path to samtools on the computer (default: "samtools') (str)
- **base_quality_limit** – base quality limit for the pileup generation (default: 30) (int)
- **samtools_flags** – additional flags for samtools (default: ' -B -d ' + str(SAMTOOLS_MAX_DEPTH) + ' ') (str)

Returns df_joined: A dataframe containing the detailed results. (pandas.DataFrame)

`isomut2py.postprocess.`**`optimize_results`**(*sample_names*, *control_samples*, *FPs_per_genome*, *plot_roc=False*, *plot_tuning_curve=False*, *filtered_results_file=None*, *output_dir=None*, *mutations_dataframe=None*)

Optimizes the list of detected mutations according to the list of control samples and desired level of false positives set by the user. Filtered results will be loaded to the mutations attribute of the MutationDetection object.

Parameters

- **sample_names** – The list of sample names included in the analysis. (list of str)
- **control_samples** – List of sample names that should be used as control samples in the sense, that no unique mutations are expected in them. (The sample names listed here must match a subset of the sample names listed in bam_filename.) (list of str)
- **FPs_per_genome** – The total number of false positives tolerated in a control sample. (int)

- **plot_roc** – If True, ROC curves will be plotted as a visual representation of the optimization process. (default: False) (boolean)
- **plot_tuning_curve** – If True, tuning curves displaying the number of mutations found in different samples with different score filters will be plotted as a visual representation of the optimization process. (default: False) (boolean)
- **filtered_results_file** – The path to the file where filtered results should be saved. (default: [output_dir]/filtered_results.csv) (str)
- **output_dir** – the path to the directory where raw mutation tables are located (default: None) (str)
- **mutations_dataframe** – the pandas.DataFrame where mutations are located (default: None) (pandas.DataFrame)

Returns

(score_lim_dict, filtered_results)

- score_lim_dict: a dictionary containing the optimized score values for each ploidy separately
- filtered_results: a pandas.DataFrame containing the filtered mutations

`isomut2py.postprocess.`**`optimize_score`**(*mutation_dataframe*, *control_samples*, *FPs_per_genome*, *score0=0*, *unique_samples=None*)

Optimizes score values for different mutation types (SNV, INS, DEL) and ploidies according to the list of control samples and the desired level of false positives in the genome. The results are stored in the score_lim_dict attribute of the MutationDetection object. If plot = True, plots ROC curves for all mutations types (SNV, INS, DEL) and all ploidies.

Parameters

- **mutation_dataframe** – The dataframe containing the mutations. (pandas.DataFrame)
- **control_samples** – a subset of bam_filename (list of sample names) that should be considered as control samples. Control samples are defined as samples where no unique mutations are expected to be found. (list of str)
- **FPs_per_genome** – the largest number of false positives tolerated in a control sample (int)
- **score0** – Score optimization starts with score0. If a larger score value is likely to be optimal, setting score0 to a number larger than 0 can decrease computation time. (default: 0) (float)
- **unique_samples** – list of unique samples where at least one mutation is detected (default: None) (list of str)

Returns a dictionary containing the optimized score values for each ploidy

7.5 Technical (formatting, IO, etc.) functions

Most of these functions can be called as class methods for *MutationCaller objects* and/or *PloidyEstimator objects*. They are described here individually, so that subtasks can be more easily managed.

7.5.1 Formatting functions

`isomut2py.format.`**`generate_ploidy_info_file`**(*filename=None, sample_names=None, bed_filepaths=None, ploidy_estimation_objects=None, sample_groups=None, group_bed_filepaths=None*)

Generate ploidy info file for mutation detection in samples with nondefault ploidies. Make sure to supply one of the follow

- ploidy_estimation_objects
- sample_names AND bed_filepaths
- sample_groups AND group_bed_filepaths

Parameters

- **`filename`** – The desired path to the generated ploidy info file. If None, ploidy information is saved to ploidy_info_file.txt in the current directory. (default: None) (str)
- **`sample_names`** – List of bam filenames for the samples, must be supplied together with bed_filepaths. (default: None) (list of str)
- **`bed_filepaths`** – Must be supplied together with sample_names. The list of filepaths to each bed file describing the given sample in sample_names. (default: None) (list of str)
- **`ploidy_estimation_objects`** – List of PloidyEstimation objects for each sample. (default: None) (list of isomut2py.PloidyEstimation)
- **`sample_groups`** – List of lists of str. Each list in sample_groups must contain the name of bam files in that group. Must be supplied together with group_bed_filepaths. (default: None) (list of list of str, example: [[‘sample1.bam’, ‘sample2.bam’, ‘sample3.bam’], [‘sample4.bam’, ‘sample5.bam’], [‘sample6.bam’]])
- **`group_bed_filepaths`** – List of filepaths to the bed files describing each sample group in sample_groups. Must be supplied together with sample_groups. (default: None) (list of str, example: [‘bedfile_of_samples123.txt’, ‘bedfile_of_samples45.txt’ ‘bedfile_of_samples6.txt’])

`isomut2py.format.`**`get_bed_format_for_sample`**(*chromosomes, chrom_length, output_dir, bam_filename=None, ownbed_filepath=None*)

Creates bed file of constant ploidies for a given sample from a file of positional ploidy data. If the ownbed_filepath attribute of the PloidyEstimation object is set, saves the bedfile to the path specified there. Otherwise, saves it to the output_dir with the “_ploidy.bed” suffix. Also sets the bed_dataframe attribute to the pandas.Dataframe containing the bed file.

Parameters

- **`chromosomes`** – list of chromosomes (array-like)
- **`chrom_length`** – list of chromosome lengths in bp (array-like)
- **`output_dir`** – path to the directory where the PE_fullchrom_* files are located. (str)
- **`bam_filename`** – filename of the BAM file of the sample (default: None) (str)
- **`ownbed_filepath`** – path to the bed file where results should be saved (default: None) (str)

Returns

(ownbed_filepath, df)

- ownbed_filepath: path the the bed file where results are saved
- df: the bed file in a pandas.DataFrame

7.5.2 IO functions

`isomut2py.io.`**`get_coverage_distribution`**(*chromosomes*, *output_dir*, *cov_max*, *cov_min*)

Sets the coverage_sample attribute of the PloidyEstimation object to the coverage distribution obtained from the temporary files created by __PE_prepare_temp_files(). Positions are filtered according to the attributes of the PloidyEstimation object. The number of positions in the final sample is decreased to 2000 for faster inference.

Parameters

- **`chromosomes`** – list of chromosomes in the samples (array-like)
- **`output_dir`** – path to the directory where temporary files (PEtmp_fullchrom*) are located (str)
- **`cov_max`** – the maximum value of the coverage for a position to be included in the analysis (int)
- **`cov_min`** – the minimum value of the coverage for a position to be included in the analysis (int)

Returns a 2000-element sample from the coverage distribution

`isomut2py.io.`**`load_bedfile_from_file`**(*filename=None*, *output_dir=None*, *bam_filename=None*)

Loads the bedfile containing previous ploidy estimated for the given sample from the path specified in filename.

Parameters

- **`filename`** – The path to the bedfile. If not supplied, the bed file is attempted to be loaded from [output_dir]/[bam_filename]_ploidy.bed. (default: None) (str)
- **`output_dir`** – The path to the directory where the default bedfile is located. (default: None) (str)
- **`bam_filename`** – the filename of the BAM file of the sample (default: None) (str)

Returns the pandas DataFrame of the bedfile

`isomut2py.io.`**`load_cov_distribution_parameters_from_file`**(*filename=None*, *output_dir=None*)

Loads the parameters of the seven fitted Gaussians to the coverage distribution of the sample from the specified filename (that was saved with pickle beforehand). If one such file is available, the computationally expensive ploidy estimation process can be skipped.

Parameters

- **`filename`** – The path to the file with the coverage distribution parameters. (default: None) (str)
- **`output_dir`** – if filename is not supplied, distribution parameters are attempted to be loaded from [output_dir]/GaussDistParams.pkl (default: None) (str)

Returns dictionary containing the fitted coverage distribution parameters

`isomut2py.io.`**`load_mutations`**(*output_dir*, *filename=None*)

Loads mutations from a file or a list of files.

Parameters

- **`filename`** – The path to the file, where mutations are stored. A list of paths can be also supplied, in this case, all of them will be loaded to a single dataframe. If None, the file [output_dir]/filtered_mutations.csv will be loaded. (default: None) (str)
- **`output_dir`** – The path to the directory containing the file filtered_results.csv. (str)

Returns a pandas.DataFrame containing the mutations

`isomut2py.io.`**`load_obj`**(*name*)

Loads a python object from a file with the name (name).pkl.

Parameters **`name`** – filename (without extension)

Returns loaded object

`isomut2py.io.`**`save_obj`**(*obj*, *name*)

Saves a python object (obj) with the specified name as (name).pkl.

Parameters

- **`obj`** – object to save
- **`name`** – filename (without extension)

7.5.3 Processing functions

(These functions mainly take care of parallelization.)

`isomut2py.process.`**`PE_prepare_temp_files`**(*ref_fasta*, *input_dir*, *bam_filename*, *output_dir*, *genome_length*, *n_min_block*, *n_conc_blocks*, *chromosomes*, *chrom_length*, *windowsize=10000*, *shiftsize=3000*, *min_noise=0.1*, *print_every_nth=1000*, *base_quality_limit=0*, *samtools_fullpath='samtools'*, *samtools_flags=None*, *bedfile=None*, *level=0*)

Prepares temporary files for ploidy estimation, by averaging coverage in moving windows for the whole genome and collecting positions with reference allele frequencies in the [min_noise, 1-min_noise] range.

Parameters

- **`ref_fasta`** – path to the reference genome fasta file (str)
- **`input_dir`** – path to the folder where bam files are located (str)
- **`bam_filename`** – list of bam filenames to analyse (list of str)
- **`output_dir`** – path to the directory where results should be saved (str)
- **`genome_length`** – the total length of the genome in basepairs (int)
- **`n_min_block`** – The approximate number of blocks to partition the analysed genome to for parallel computing. The actual number might be slightly larger that this. (default: 200) (int)
- **`n_conc_blocks`** – The number of blocks to process at the same time. (default: 4) (int)
- **`chromosomes`** – list of chromosomes in the genome (list of str)
- **`chrom_length`** – list of chromosome lengths in the genome (list of int)
- **`windowsize`** – windowsize for initial coverage smoothing with a moving average method (default: 10000) (int)

- **shiftsize** – shiftsize for initial coverage smoothing with a moving average method (default: 3000) (int)
- **min_noise** – The minimum frequency of non-reference or reference bases detected for a position to be considered for LOH detection. Setting it too small will result in poor noise filtering, setting it too large will result in a decreased number of measurement points. (default: 0.1) (float in range(0,1))
- **print_every_nth** – Even though LOH detection is limited to the positions with a noise level larger that min_noise, ploidy estimation is based on all the genomic positions meeting the above set criteria. By setting the attribute print_every_nth, the number of positions used can be controlled. Setting it large will result in overlooking ploidy variations in shorter genomic ranges, while setting it too small can cause an increase in both memory usage and computation time. Decrease only if a relatively short genome is analysed. (default: 100) (int)
- **base_quality_limit** – The base quality limit used by samtools in order to decide if a base should be included in the pileup file. (default: 0) (int)
- **samtools_fullpath** – The path to samtools on the computer. (default: "samtools") (str)
- **samtools_flags** – The samtools flags to be used for pileup file generation. (default: " -B -d 1000 ") (str)
- **bedfile** – path to the the bedfile to limit ploidy estimation to a specific region of the genome (default: None) (str)
- **level** – the level of indentation used in verbose output (default: 0) (int)

isomut2py.process.**double_run**(*mutcallObject*, *level=0*)

Runs the mutation detection pipeline on the MutationCaller object when with local realignment.

Parameters

- **mutcallObject** – an isomut2py.MutationCaller object
- **level** – the level of indentation used in verbose output (default: 0) (int)

isomut2py.process.**single_run**(*mutcallObject*, *level=0*)

Runs the mutation detection pipeline on the MutationCaller object when no local realignment is used during the process.

Parameters

- **mutcallObject** – an isomut2py.MutationCaller object
- **level** – the level of indentation used in verbose output (default: 0) (int)

7.5.4 Bayesian inference functions

(These functions are called by the *PloidyEstimator object* to fit different theoretical distributions to the actual coverage distribution calculated from the data.)

isomut2py.bayesian.**PE_on_chrom**(*chrom*, *output_dir*, *windowsize_PE*, *shiftsize_PE*, *distribution_dict*)

Runs the whole ploidy estimation pipeline on a given chromosome, using the appropriate attributes of the PloidyEstimation object by running PE_on_range() multiple times. Prints the results to the file: [output_dir]/PE_fullchrom_[chrom].txt.

Parameters

- **`distribution_dict`** – a dictionary containing the fitted parameters of the Gaussian mixture model to the coverage distribution. (dict with keys 'mu', 'sigma' and 'p')
- **`shiftsize_PE`** – shiftsize for moving average over regions (int)
- **`windowsize_PE`** – windowsize for moving average over regions (int)
- **`output_dir`** – the path to the directory where temporary files are located (str)
- **`chrom`** – the name of the chromosome to analyse (str)

`isomut2py.bayesian.`**`PE_on_range`**(*dataframe*, *rmin*, *rmax*, *all_mu*, *all_sigma*, *prior*, *cov_min=0*, *cov_max=100000*)

Run the ploidy estimation on a given range of a chromosome.

Parameters

- **`dataframe`** – The dataframe read from the temporary files for a given chromosome, containing information about the genomic position, the measured coverage and the frequency of the non-reference bases aligned to the position. (pandas.DataFrame)
- **`rmin`** – The lower bound of the genomic range considered for the analysis. (int)
- **`rmax`** – The upper bound of the genomic range considered for the analysis. (int)
- **`all_mu`** – The centers of the seven Gaussians fitted to the coverage distribution. (list of float)
- **`all_sigma`** – The sigmas of the seven Gaussians fitted to the coverage distribution. (list of float)
- **`prior`** – The weights of the seven Gaussians fitted to the coverage distribution. (list of float in the range(0,1))

Returns

(most_probable_ploidy, most_probable_LOH)

- most_probable_ploidy: the estimated ploidy for the genomic region (int, in range(1,8))
- most_probable_LOH: the estimated LOH status for the genomic region (1: LOH, 0: no LOH)

`isomut2py.bayesian.`**`estimate_hapcov_infmix`**(*cov_min=None*, *hc_percentile=None*, *chromosomes=None*, *output_dir=None*, *cov_max=None*, *level=0*, *cov_sample=None*, *user_defined_hapcov=None*)

Estimates the haploid coverage of the sample. If the user_defined_hapcov attribute is set manually, it sets the value of estimated_hapcov to that. Otherwise, a many-component (20) Gaussian mixture model is fitted to the coverage histogram of the sample 10 times. Each time, the haploid coverage is estimated from the center of the component with the maximal weight in the model. The final estimate of the haploid coverage is calculated as the qth percentile of the 10 measurements, with q = hc_percentile.

Parameters

- **`user_defined_hapcov`** – if not None, its value is returned (default: None) (float)
- **`cov_sample`** – a sample of the coverage distribution of the investigated sample, if None, it is loaded from the temporary files of the output_dir (default: None) (array-like)
- **`cov_max`** – the maximum value of the coverage for a position to be considered in the estimation (default: None) (int)
- **`output_dir`** – the path to the output directory of the PloidyEstimator object, where temporary files are located (default: None) (str)

- **`chromosomes`** – list of chromosomes for the sample (default: None) (array-like)
- **`hc_percentile`** – the percentile value to use for calculating the estimated haploid coverage from 10 subsequent estimations (default: None) (int)
- **`cov_min`** – the maximum value of the coverage for a position to be considered in the estimation (default: None) (int)
- **`level`** – the level of indentation used in verbose output (default: 0) (int)

Returns (estimated haploid coverage (float), sample from coverage distribution (array-like))

`isomut2py.bayesian.`**`fit_gaussians`**(*estimated_hapcov*, *chromosomes=None*, *output_dir=None*, *cov_max=None*, *cov_min=None*, *level=0*, *cov_sample=None*)

Fits a 7-component Gaussian mixture model to the coverage distribution of the sample, using the appropriate attributes of the PloidyEstimation object. The center of the first Gaussian is initialized from a narrow region around the value of the estimated_hapcov attribute. The centers of the other Gaussians are initialized in a region around the value of estimated_hapcov multiplied by consecutive whole numbers.

The parameters of the fitted model (center, sigma and weight) for all seven Gaussians are both saved to the GaussDistParams.pkl file (in output_dir, for later reuse) and set as the value of the distribution_dict attribute.

Parameters

- **`cov_sample`** – a sample of the coverage distribution of the investigated sample, if None, it is loaded from the temporary files of the output_dir (default: None) (array-like)
- **`cov_min`** – the maximum value of the coverage for a position to be considered in the estimation (default: None) (int)
- **`output_dir`** – the path to the output directory of the PloidyEstimator object, where temporary files are located. If not None, distribution parameters are saved there as GaussDistParams.pkl. (default: None) (str)
- **`chromosomes`** – list of chromosomes for the sample (default: None) (array-like)
- **`estimated_hapcov`** – the estimated value for the haploid coverage, used as prior (float)
- **`level`** – the level of indentation used in verbose output (default: 0) (int)

Returns dictionary containing the fitted parameters of the 7 Gaussians

7.5.5 Functions for ploidy comparison

(These functions perform the comparison of ploidy estimates of two samples for different file formats.)

`isomut2py.compare.`**`check_interval_for_difference`**(*chrom*, *chromStart*, *chromEnd*, *ploidy1*, *ploidy2*, *original_bamfile1*, *original_bamfile2*, *refgenome*, *prior_dist_dict1*, *prior_dist_dict2*)

Checks if a genomic interval with different estimated ploidies is really different in the two samples.

Parameters

- **`chrom`** – The chromosome of the genomic interval. (str)
- **`chromStart`** – The starting position of the genomic interval. (int)
- **`chromEnd`** – The ending position of the genomic interval. (int)
- **`ploidy1`** – The ploidy estimated for the genomic interval for original_bamfile1. (int)
- **`ploidy2`** – The ploidy estimated for the genomic interval for original_bamfile2. (int)

- **`original_bamfile1`** – The path to the original bamfile containing alignment information for sample1. (str)
- **`original_bamfile2`** – The path to the original bamfile containing alignment information for sample2. (str)
- **`refgenome`** – The path to the reference genome fasta file. (str)
- **`prior_dist_dict1`** – The parameters of the seven Gaussians fitted to the coverage distribution of sample1. (dict)
- **`prior_dist_dict2`** – The parameters of the seven Gaussians fitted to the coverage distribution of sample2. (dict)

Returns the ratio of the likelihood of the two samples having different ploidies in the region and the likelihood of them having the same ploidies (float) - if the ratio of them having different ploidies is smaller than the ratio of them having the same one, the returned value is 0

`isomut2py.compare.`**`compare_with_bed`**(*bed_dataframe*, *other_file*, *minLen*)

Compares the results of ploidy estimation with a bed file defined in other_file.

Parameters

- **`bed_dataframe`** – a pandas.DataFrame of the bedfile of the sample (pandas.DataFrame)
- **`other_file`** – The path to the bedfile of the other sample. (str)
- **`minLen`** – The minimum length of a region to be considered different from the other_file. (int)

Returns df_joined: A pandas.DataFrame containing region information from both the PloidyEstimation object and the other_file.

`isomut2py.compare.`**`compare_with_other_PloidyEstimator`**(*ob1*, *ob2*, *minLen*, *minQual*)

Compare the estimated ploidies of the PloidyEstimation object with the ploidies of another PloidyEstimator object.

Parameters

- **`ob2`** – the other PloidyEstimator object (isomut2py.ploidyestimation.PloidyEstimator)
- **`ob1`** – the first PloidyEstimator object (isomut2py.ploidyestimation.PloidyEstimator)
- **`minLen`** – The minimum length of a region to be considered different from the other object. (int)
- **`minQual`** – The minimum quality of a region to be considered different from the other object. (float)

Returns df_intervals: The differing intervals meeting the above criteria. (pandas.DataFrame)

7.5.6 Functions for loading example parameter settings

(These functions download example datasets and help load the settings for processing these in a concise way.)

`isomut2py.examples.`**`download_example_data`**(*path='.'*)

Download example data from http://genomics.hu/tools/isomut2py/isomut2py_exampleDataset.tar.gz to path.

Parameters **`path`** – where to download (default: '.') (str)

`isomut2py.examples.`**`download_raw_example_data`**(*path='.'*)

Download raw example data from either http://genomics.hu/tools/isomut2py/isomut2py_rawExampleDataset.tar.gz or http://genomics.hu/tools/isomut2py/isomut2py_rawExampleDataset_shortgenome.tar.gz to path.

Parameters **`path`** – where to download (default: '.') (str)

`isomut2py.examples.`**`load_example_mutdet_settings`**(*example_data_path='.'*, *output_dir='.'*)

Loads example settings for mutation detection.

Parameters

- **`example_data_path`** – the path where example data is located (default: '.') (str)
- **`output_dir`** – the path where results should be saved (default: current working directory) (str)

Returns dictionary containing example parameters

`isomut2py.examples.`**`load_example_ploidyest_settings`**(*example_data_path='.'*, *output_dir='.'*)

Loads example settings for ploidy estimation.

Parameters

- **`example_data_path`** – the path where example data is located (default: '.') (str)
- **`output_dir`** – the path where results should be saved (default: current working directory) (str)

Returns dictionary containing example parameters

`isomut2py.examples.`**`load_preprocessed_example_ploidyest_settings`**(*example_data_path='.'*)

Loads example settings for ploidy estimation with preprocessed files.

Parameters **`example_data_path`** – the path where example data is located (default: '.') (str)

Returns dictionary containing example parameters

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index