CNVkit Documentation

Release 0.6.1

Eric Talevich

August 14, 2015

Contents

Author Eric Talevich

Contact eric.talevich@ucsf.edu

License Apache License 2.0

Source code GitHub

Packages PyPI | Docker | Galaxy | DNAnexus

Q&A Biostars | SeqAnswers

CNVkit is a Python library and command-line software toolkit to infer and visualize copy number from targeted DNA sequencing data. It is designed for use with hybrid capture, including both whole-exome and custom target panels, and short-read sequencing platforms such as Illumina and Ion Torrent.

CHAPTER 1

Quick start

If you would like to quickly try CNVkit without installing it, try our app on DNAnexus.

To run CNVkit on your own machine, keep reading.

1.1 Install CNVkit

Download the source code from GitHub:

http://github.com/etal/cnvkit

And read the README file.

1.2 Download the reference genome

Go to the UCSC Genome Bioinformatics website and download:

1. Your species' reference genome sequence, in FASTA format [required]
2. Gene annotation database, via RefSeq or Ensembl, in "flat" format (e.g. refFlat.txt) [optional]

You probably already have the reference genome sequence. If your species' genome is not available from UCSC, use whatever reference sequence you have. CNVkit only requires that your reference genome sequence be in FASTA format. Both the reference genome sequence and the annotation database must be single, uncompressed files.

Sequencing-accessible regions: If your reference genome is the UCSC human genome hg19, a BED file of the sequencing-accessible regions is included in the CNVkit distribution as `data/access-5kb-mappable.hg19.bed`. If you're not using hg19, consider building the "access" file yourself from your reference genome sequence (say, `mm10.fasta`) using the bundled script `genome2access.py`:

```
genome2access.py mm10.fasta -s 10000 -o access-10kb.mm10.bed
```

We'll use this file in the next step to ensure off-target bins ("antitargets") are allocated only in chromosomal regions that can be mapped.

Gene annotations: The gene annotations file (refFlat.txt) is useful to apply gene names to your baits BED file, if the BED file does not already have short, informative names for each bait interval. This file can be used in the next step.

If your targets look like:

```
chr1        1508981 1509154
chr1        2407978 2408183
chr1        2409866 2410095
```

Then you want refFlat.txt.

Otherwise, if they look like:

```
chr1        1508981 1509154 SSU72
chr1        2407978 2408183 PLCH2
chr1        2409866 2410095 PLCH2
```

Then you don't need refFlat.txt.

1.3 Map sequencing reads to the reference genome

If you haven't done so already, use a sequence mapping/alignment program such as BWA to map your sequencing reads to the reference genome sequence.

You should now have one or BAM files corresponding to individual samples.

1.4 Build a reference from normal samples and infer tumor copy ratios

Here we'll assume the BAM files are a collection of "tumor" and "normal" samples, although germline disease samples can be used equally well in place of tumor samples.

CNVkit uses the bait BED file (provided by the vendor of your capture kit), reference genome sequence, and sequencing-accessible regions along with your BAM files to:

1. Create a pooled reference of per-bin copy number estimates from several normal samples; then
2. Use this reference in processing all tumor samples that were sequenced with the same platform and library prep.

All of these steps are automated with the *batch* command. Assuming normal samples share the suffix "Normal.bam" and tumor samples "Tumor.bam", a complete command could be:

```
cnvkit.py batch *Tumor.bam --normal *Normal.bam \
    --targets my_targets.bed --fasta hg19.fasta \
    --split --access data/access-5kb-mappable.hg19.bed \
    --output-reference my_reference.cnn --output-dir example/
```

See the built-in help message to see what these options do, and for additional options:

```
cnvkit.py batch -h
```

If you have no normal samples to use for the reference, you can create a "flat" reference which assumes equal coverage in all bins by using the `--normal/-n` flag without specifying any additional BAM files:

```
cnvkit.py batch *Tumor.bam -n -t my_targets.bed -f hg19.fasta \
    --split --access data/access-5kb-mappable.hg19.bed \
    --output-reference my_flat_reference.cnn -d example2/
```

In either case, you should run this command with the reference genome sequence FASTA file to extract GC and RepeatMasker information for bias corrections, which enables CNVkit to improve the copy ratio estimates even without a paired normal sample.

If your targets are missing gene names, you can add them here with the `--annotate` argument:

```
cnvkit.py batch *Tumor.bam -n *Normal.bam -t my_targets.bed -f hg19.fasta \
    --annotate refFlat.txt --split --access data/access-5kb-mappable.hg19.bed \
    --output-reference my_flat_reference.cnn -d example3/
```

1.5 Process more tumor samples

You can reuse the reference file you've previously constructed to extract copy number information from additional tumor sample BAM files, without repeating the steps above. Assuming the new tumor samples share the suffix "Tumor.bam" (and let's also spread the workload across all available CPUs with the `-p` option, and generate some figures):

```
cnvkit.py batch *Tumor.bam -r my_reference.cnn -p 0 --scatter --diagram -d example4/
```

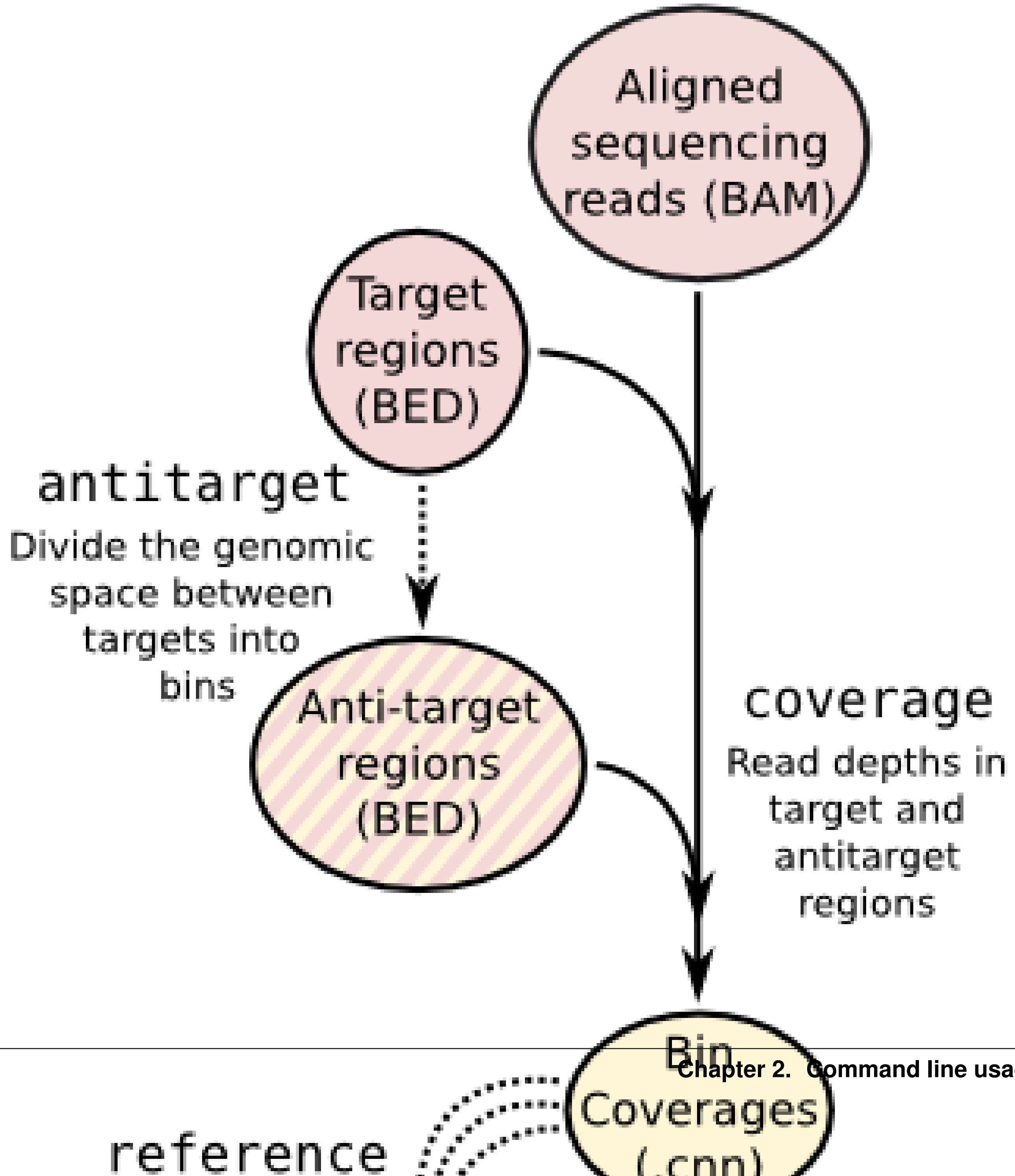
The coordinates of the target and antitarget bins, the gene names for the targets, and the GC and RepeatMasker information for bias corrections are automatically extracted from the reference .cnn file you've built.

See the command-line usage pages for additional visualization, reporting and import/export commands in CNVkit.

CHAPTER 2

Command line usage

2.1 Copy number calling pipeline


Aligned sequencing reads (BAM)
Target regions (BED)
antitarget
Divide the genomic space between targets into bins
Anti-target regions (BED)
coverage
Read depths in target and antitarget regions
Bin Coverages (.cnn)
reference

Each operation is invoked as a sub-command of the main script, `cnvkit.py`. A listing of all sub-commands can be obtained with `cnvkit --help` or `-h`, and the usage information for each sub-command can be shown with the `--help` or `-h` option after each sub-command name:

```
cnvkit.py -h
cnvkit.py antitarget -h
```

A sensible output file name is normally chosen if it isn't specified, except in the case of the text reporting commands, which print to standard output by default, and the matplotlib-based plotting commands (not `diagram`), which will display the plots interactively on the screen by default.

2.1.1 batch

Run the CNVkit pipeline on one or more BAM files:

```
cnvkit.py batch Sample.bam -t Tiled.bed -a Background.bed -r Reference.cnn
cnvkit.py batch *.bam --output-dir CNVs/ -t Tiled.bed -a Background.bed -r Reference.cnn
```

With the `-p` option, process each of the BAM files in parallel, as separate subprocesses. The status messages logged to the console will be somewhat disorderly, but the pipeline will take advantage of multiple CPU cores to complete sooner.

```
cnvkit.py batch *.bam -d CNVs/ -t Tiled.bed -a Background.bed -r Reference.cnn -p 8
```

The pipeline executed by the `batch` command is equivalent to:

```
cnvkit.py coverage Sample.bam Tiled.bed -o Sample.targetcoverage.cnn
cnvkit.py coverage Sample.bam Background.bed -o Sample.antitargetcoverage.cnn
cnvkit.py fix Sample.targetcoverage.cnn Sample.antitargetcoverage.cnn Reference_cnn -o Sample.cnr
cnvkit.py segment Sample.cnr -o Sample.cns
```

See the rest of the commands below to learn about each of these steps and other functionality in CNVkit.

2.1.2 target

Prepare a BED file of baited regions for use with CNVkit.

```
cnvkit.py target Tiled.bed --annotate refFlat.txt --split -o Targets.bed
```

The BED file should be the baited genomic regions for your target capture kit, as provided by your vendor. Since these regions (usually exons) may be of unequal size, the `--split` option divides the larger regions so that the average bin size after dividing is close to the size specified by `--average-size`.

In case the vendor BED file does not label each region with a corresponding gene name, the `--annotate` option can add or replace these labels. Gene annotation databases, e.g. RefSeq or Ensembl, are available in "flat" format from UCSC (e.g. refFlat.txt for hg19).

In other cases the region labels are a combination of human-readable gene names and database accession codes, separated by commas (e.g. "ref|BRAF,mRNA|AB529216,ens|ENST00000496384"). The `--short-names` option splits these accessions on commas, then chooses the single accession that covers in the maximum number of consecutive regions that share that accession, and applies it as the new label for those regions. (You may find it simpler to just apply the refFlat annotations.)

If you need higher resolution, you can select a smaller average size for your *target* and *antitarget* bins.

Exons in the human genome have an average size of about 200bp. The target bin size default of 267 is chosen so that splitting larger exons will produce bins with a minimum size of 200. Since bins that contain fewer reads result in a noisier copy number signal, this approach ensures the “noisiness” of the bins produced by splitting larger exons will be no worse than average.

Setting the average size of target bins to 100bp, for example, will yield about twice as many target bins, which can result in more precise and perhaps more accurate segmentation. However, the number of reads counted in each bin will be reduced by about half, increasing the variance or “noise” in bin-level coverages. An excess of noisy bins can make visualization difficult, and since the noise may not be Gaussian, especially in the presence of many bins with zero reads, the CBS algorithm could produce less accurate segmentation results on low-coverage samples. In practice we see good results with an average of 200-300 reads per bin; we therefore recommend an overall on-target sequencing coverage depth of at least 200x to 300x with a read length of 100 to justify reducing the average target bin size to 100bp.

2.1.3 antitarget

Given a “target” BED file that lists the chromosomal coordinates of the tiled regions used for targeted resequencing, derive a BED file off-target/”antitarget”/”background” regions.

```
cnvkit.py antitarget Tiled.bed -g data/access-5kb-mappable.hg19.bed -o Background.bed
```

Many fully sequenced genomes, including the human genome, contain large regions of DNA that are inaccessable to sequencing. (These are mainly the centromeres, telomeres, and highly repetitive regions.) In the FASTA genome sequence these regions are filled in with large stretches of “N” characters. These regions cannot be mapped by resequencing, so we can avoid them when calculating the antitarget locations by passing the locations of the accessible sequence regions with the `-g` or `--access` option. These regions are precomputed for the UCSC reference human genome hg19 (data/access-5kb-mappable.hg19.bed), and can be computed for other genomes with the included script `genome2access.py`.

CNVkit uses a cautious default off-target bin size that, in our experience, will typically include more reads than the average on-target bin. However, we encourage the user to examine the coverage statistics reported by CNVkit and specify a properly calculated off-target bin size for their samples in order to maximize copy number information.

2.1.4 coverage

Calculate coverage in the given regions from BAM read depths.

With the -p option, calculates mean read depth from a pileup; otherwise, counts the number of read start positions in the interval and normalizes to the interval size.

```
cnvkit.py coverage Sample.bam Tiled.bed -o Sample.targetcoverage.cnn
cnvkit.py coverage Sample.bam Background.bed -o Sample.antitargetcoverage.cnn
```

Summary statistics of read counts and their binning are printed to standard error when CNVkit finishes calculating the coverage of each sample (through either the *batch* or *coverage* commands).

Note: The BAM file must be sorted. CNVkit (and most other software) will not notice out if the reads are out of order; it will just ignore the out-of-order reads and the coverages will be zero after a certain point early in the file (e.g. in the middle of chromosome 2). A future release may try to be smarter about this.

Note: If you’ve prebuilt the BAM index file (.bai), make sure its timestamp is later than the BAM file’s. CNVkit will automatically index the BAM file if needed – that is, if the .bai file is missing, *or* if the timestamp of the .bai file is older than that of the corresponding .bam file. This is done in case the BAM file has changed after the index was initially created. (If the index is wrong, CNVkit will not catch this, and coverages will be mysteriously truncated to zero after a certain point.) *However,* if you copy a set of BAM files and their index files (.bai) together over a network, the smaller .bai files will typically finish downloading first, and so their timestamp will be earlier than the corresponding BAM or FASTA file. CNVkit will then consider the index files to be out of date and will attempt to rebuild them. To prevent this, use the Unix command `touch` to update the timestamp on the index files after all files have been downloaded.

2.1.5 reference

Compile a copy-number reference from the given files or directory (containing normal samples). If given a reference genome (-f option), also calculate the GC content of each region.

```
cnvkit.py reference -o Reference.cnn -f ucsc.hg19.fa *targetcoverage.cnn
```

If normal samples are not available, it will sometimes work OK to build the reference from a collection of tumor samples. You can use the `scatter` command on the raw `.cnn` coverage files to help choose samples with relatively minimal CNVs for use in the reference.

Notes on sample selection:

- You can use `cnvkit.py metrics *.cnr -s *.cns` to see if any samples are especially noisy. See the *metrics* command.
- CNVkit will usually call larger CNAs reliably down to about 10x on-target coverage, but there will tend to be more spurious segments, and smaller-scale or subclonal CNAs can be hard to infer below that point. This is well below the minimum coverage thresholds typically used for SNV calling, especially for targeted sequencing of tumor samples that may have significant normal-cell contamination and subclonal tumor-cell populations. So, a normal sample that passes your other QC checks will probably be OK to use in building a CNVkit reference – assuming it was sequenced on the same platform as the other samples you're calling.

Alternatively, you can create a "flat" reference of neutral copy number (i.e. log2 0.0) for each probe from the target and antitarget interval files. This still computes the GC content of each region if the reference genome is given.

```
cnvkit.py reference -o FlatReference.cnn -f ucsc.hg19.fa -t Tiled.bed -a Background.bed
```

Two possible uses for a flat reference:

1. Extract copy number information from one or a small number of tumor samples when no suitable reference or set of normal samples is available. The copy number calls will not be as accurate, but large-scale CNVs may still be visible.
2. Create a "dummy" reference to use as input to the `batch` command to process a set of normal samples. Then, create a "real" reference from the resulting `*.targetcoverage.cnn` and `*.antitargetcoverage.cnn` files, and re-run `batch` on a set of tumor samples using this updated reference.

Note: As with BAM files, CNVkit will automatically index the FASTA file if the corresponding .fai file is missing or out of date. If you have copied the FASTA file and its index together over a network, you may need to use the `touch` command to update the .fai file's timestamp so that CNVkit will recognize it as up-to-date.

2.1.6 fix

Combine the uncorrected target and antitarget coverage tables (.cnn) and correct for biases in regional coverage and GC content, according to the given reference. Output a table of copy number ratios (.cnr).

```
cnvkit.py fix Sample.targetcoverage.cnn Sample.antitargetcoverage.cnn Reference.cnn -o Sample.cnr
```

2.1.7 segment

Infer discrete copy number segments from the given coverage table:

```
cnvkit.py segment Sample.cnr -o Sample.cns
```

By default this uses the circular binary segmentation algorithm (CBS), but with the `-m` option, the faster Fused Lasso algorithm (`flasso`) or even faster but less accurate HaarSeg algorithm (`haar`) can be used instead.

Fused Lasso additionally performs significance testing to distinguish CNAs from regions of neutral copy number, whereas CBS and HaarSeg by themselves only identify the supported segmentation breakpoints.

2.1.8 call

Given segmented log2 ratio estimates (.cns), round the copy ratio estimates to integer values using either:

- A list of threshold log2 values for each copy number state, or
- Some algebra, given known tumor cell fraction and normal ploidy.

```
cnvkit.py call Sample.cns -o Sample.call.cns
cnvkit.py call Sample.cns -y -m clonal --purity 0.65 -o Sample.call-clonal.cns
cnvkit.py call Sample.cns -y -m threshold -t=-1.1,-0.4,0.3,0.7 -o Sample.call-threshold.cns
```

The output is another .cns file, where the values in the log2 column are still log2-transformed, but represent integers in log2 scale – e.g. a neutral diploid state is represented as 0.0, not the integer 2. These output files are still compatible with the other CNVkit commands that accept .cns files, and can be plotted the same way.

The "clonal" method considers the observed log2 ratios in the input .cns file as a mix of some fraction of tumor cells (specified by `--purity`), possibly with altered copy number, and a remainder of normal cells with neutral copy number (specified by `--ploidy` for autosomes). This equation is rearranged to find the absolute copy number of the tumor cells alone, rounded to the nearest integer. The expected and observed ploidy of the sex chromosomes (X and Y) is different, so it's important to specify `-y`/`--male-reference` if a male reference was used; the sample gender can be specified if known, otherwise it will be guessed from the log2 ratio of chromosome X.

The "threshold" method simply applies fixed log2 ratio cutoff values for each integer copy number state. This method therefore does not require the tumor cell fraction or purity to be known. The default cutoffs are reasonable for a tumor sample with purity of at least 40% or so; for germline samples, the `-t` values shown above may yield more accurate calls.

2.2 Plots and graphics

2.2.1 scatter

Plot bin-level log2 coverages and segmentation calls together. Without any further arguments, this plots the genome-wide copy number in a form familiar to those who have used array CGH.

```
cnvkit.py scatter Sample.cnr -s Sample.cns
# Shell shorthane
cnvkit.py scatter -s Sample.cn{s,r}
```

The options `--chromosome` and `--gene` (or their single-letter equivalents) focus the plot on the specified region:

```
cnvkit.py scatter -s Sample.cn{s,r} -c chr7
cnvkit.py scatter -s Sample.cn{s,r} -c chr7:140434347-140624540
cnvkit.py scatter -s Sample.cn{s,r} -g BRAF
```

In the latter two cases, the genes in the specified region or with the specified names will be highlighted and labeled in the plot. The `--width` (`-w`) argument determines the size of the chromosomal regions to show flanking the selected region. Note that only targeted genes can be highlighted and labeled; genes that are not included in the list of targets are not labeled in the .cnn or .cnr files and are therefore invisible to CNVkit.

To create multiple region-specific plots at once, the regions of interest can be listed in a separate file and passed to the `scattter` command with the `-l`/`--range-list` option. This is equivalent to creating the plots separately with the `-c` option and then combining the plots into a single multi-page PDF.

The arguments `-c` and `-g` can be combined to e.g. highlight specific genes in a larger context:

```
# Show the whole chromosome, highlight two genes
cnvkit.py scatter -s Sample.cn{s,r} -c chr7 -g BRAF,MET
# Show a chromosome arm, highlight one gene
cnvkit.py scatter -s Sample.cn{s,r} -c chr5:100-50000000 -g TERT
```

When a chromosomal region is plotted with CNVkit's "scatter" command , the size of the plotted datapoints is proportional to the weight of each point used in segmentation – a relatively small point indicates a less reliable bin. Therefore, if you see a cluster of smaller points in a short segment (or where you think there ought to be a segment, but there isn't one), then you can cast some doubt on the copy number call in that region. The dispersion of points around the segmentation line also visually indicates the level of noise or uncertainty.

Loss of heterozygosity (LOH) can be viewed alongside copy number by passing variants as a VCF file with the `-v` option. Heterozygous SNP allelic frequencies are shown in a subplot below the CNV scatter plot. (Also see the *loh* command.)

```
cnvkit.py scatter Sample.cnr -s Sample.cns -v Sample.vcf
```

The bin-level log2 ratios or coverages can also be plotted without segmentation calls:

```
cnvkit.py scatter Sample.cnr
```

This can be useful for viewing the raw, un-corrected coverage depths when deciding which samples to use to build a profile, or simply to see the coverages without being helped/biased by the called segments.

The `--trend` option (`-t`) adds a smoothed trendline to the plot. This is fairly superfluous if a valid segment file is given, but could be helpful if the CBS dependency is not available, or if you're skeptical of the segmentation in a region.

2.2.2 loh

Plot allelic frequencies at each variant position in a VCF file. Divergence from 0.5 indicates loss of heterozygosity (LOH) in a tumor sample.

```
cnvkit.py loh Sample.vcf
```

2.2.3 diagram

Draw copy number (either individual bins (.cnn, .cnr) or segments (.cns)) on chromosomes as an ideogram. If both the bin-level log2 ratios and segmentation calls are given, show them side-by-side on each chromosome (segments on the left side, bins on the right side).

```
cnvkit.py diagram Sample.cnr
cnvkit.py diagram -s Sample.cns
cnvkit.py diagram -s Sample.cns Sample.cnr
```

If bin-level log2 ratios are provided (.cnr), genes with log2 ratio values beyond a fixed threshold will be labeled on the plot. This plot style works best with target panels of a few hundred genes at most; with whole-exome sequencing there are often so many genes affected by CNAs that the individual gene labels become difficult to read.

If only segments are provided (`-s`), gene labels are not shown. This plot is then equivalent to the `heatmap` command, which effectively summarizes the segmented values from many samples.

2.2.4 heatmap

Draw copy number (either bins (.cnn, .cnr) or segments (.cns)) for multiple samples as a heatmap.

To get an overview of the larger-scale CNVs in a cohort, use the "heatmap" command on all .cns files:

```
cnvkit.py heatmap *.cns
```

The color range can be subtly rescaled with the `-d` option to de-emphasize low-amplitude segments, which are likely spurious CNAs:

```
cnvkit.py heatmap *.cns -d
```

A heatmap can also be drawn from bin-level log2 coverages or copy ratios (.cnn, .cnr), but this will be extremely slow at the genome-wide level. Consider doing this with a smaller number of samples and only for one chromosome at a time, using the `-c` option:

```
cnvkit.py heatmap *.cnr -c chr7  # Slow!
```

If an output file name is not specified with the `-o` option, an interactive matplotlib window will open, allowing you to select smaller regions, zoom in, and save the image as a PDF or PNG file.

2.3 Text and tabular reports

2.3.1 breaks

List the targeted genes in which a segmentation breakpoint occurs.

```
cnvkit.py breaks Sample.cnr Sample.cns
```

This helps to identify genes in which (a) an unbalanced fusion or other structural rearrangement breakpoint occured, or (b) CNV calling is simply difficult due to an inconsistent copy number signal.

The output is a text table of tab-separated values, which is amenable to further processing by scripts and standard Unix tools such as `grep`, `sort`, `cut` and `awk`.

For example, to get a list of the names of genes that contain a possible copy number breakpoint:

```
cnvkit.py breaks Sample.cnr Sample.cns | cut -f1 | sort -u > gene-breaks.txt
```

2.3.2 gainloss

Identify targeted genes with copy number gain or loss above or below a threshold.

```
cnvkit.py gainloss Sample.cnr
cnvkit.py gainloss Sample.cnr -s Sample.cns -t 0.4 -m 5 -y
```

If segments are given, the log2 ratio value reported for each gene will be the value of the segment covering the gene. Where more than one segment overlaps the gene, i.e. if the gene contains a breakpoint, each segment's value will be reported as a separate row for the same gene. If a large-scale CNA covers multiple genes, each of those genes will be listed individually.

If segments are not given, the median of the log2 ratio values of the bins within each gene will be reported as the gene's overall log2 ratio value. This mode will not attempt to identify breakpoints within genes.

The threshold (`-t`) and minimum number of bins (`-m`) options are used to control which genes are reported. For example, a threshold of .2 (the default) will report single-copy gains and losses in a completely pure tumor sample (or germline CNVs), but a lower threshold would be necessary to call somatic CNAs if significant normal-cell contamination is present. Some likely false positives can be eliminated by dropping CNVs that cover a small number of bins (e.g. with `-m 3`, genes where only 1 or 2 bins show copy number change will not be reported), at the risk of missing some true positives.

Specify the reference gender (`-y` if male) to ensure CNVs on the X and Y chromosomes are reported correctly; otherwise, a large number of spurious gains or losses on the sex chromosomes may be reported.

The output is a text table of tab-separated values, like that of *breaks*. Continuing the Unix example, we can try `gainloss` both with and without the segment files, take the intersection of those as a list of "trusted" genes, and visualize each of them with *scatter*:

```
cnvkit.py gainloss -y Sample.cnr -s Sample.cns | tail -n+2 | cut -f1 | sort > segment-gainloss.txt
cnvkit.py gainloss -y Sample.cnr | tail -n+2 | cut -f1 | sort > ratio-gainloss.txt
comm -12 ratio-gainloss.txt segment-gainloss.txt > trusted-gainloss.txt
for gene in `cat trusted-gainloss.txt`
do
    cnvkit.py scatter -s Sample.cn{s,r} -g $gene -o Sample-$gene-scatter.pdf
done
```

(The point is that it's possible.)

2.3.3 gender

Guess samples' gender from the relative coverage of chromoxsome X. A table of the sample name (derived from the filename), guessed chromosomal gender (string "Female" or "Male"), and log2 ratio value of chromosome X is printed.

```
cnvkit.py gender *.cnn *.cnr *.cns
cnvkit.py gender -y *.cnn *.cnr *.cns
```

If there is any confusion in specifying either the gender of the sample or the construction of the reference copy number profile, you can check what happened using the "gender" command. If the reference and intermediate .cnn files are available (.targetcoverage.cnn and .antitargetcoverage.cnn, which are created before most of CNVkit's corrections), CNVkit can report the reference gender and the apparent chromosome X copy number that appears in the sample:

```
cnvkit.py gender reference.cnn Sample.targetcoverage.cnn Sample.antitargetcoverage.cnn
```

The output looks like this, where columns are filename, apparent gender, and log2 ratio of chrX:

```
cnv_reference.cnn   Female  -0.176
Sample.targetcoverage.cnn   Female  -0.0818
Sample.antitargetcoverage.cnn       Female  -0.265
```

If the `-y` option was not specified when constructing the reference (e.g. `cnvkit.py batch ...`), then you have a female reference, and in the final plots you will see chrX with neutral copy number in female samples and around -1 log2 ratio in male samples.

2.3.4 metrics

Calculate the spread of bin-level copy ratios from the corresponding final segments using several statistics. These statistics help quantify how "noisy" a sample is and help to decide which samples to exclude from an analysis, or to select normal samples for a reference copy number profile.

For a single sample:

```
cnvkit.py metrics Sample.cnr -s Sample.cns
```

(Note that the order of arguments and options matters here, unlike the other commands: Everything after the `-s` flag is treated as a segment dataset.)

Multiple samples can be processed together to produce a table:

```
cnvkit.py metrics S1.cnr S2.cnr -s S1.cns S2.cns
cnvkit.py metrics *.cnr -s *.cns
```

Several bin-level log2 ratio estimates for a single sample, such as the uncorrected on- and off-target coverages and the final bin-level log2 ratios, can be compared to the same final segmentation (reusing the given segments for each coverage dataset):

```
cnvkit.py metrics Sample.targetcoverage.cnn Sample.antitargetcoverage.cnn Sample.cnr -s Sample.cns
```

In each case, given the bin-level copy ratios (.cnr) and segments (.cns) for a sample, the log2 ratio value of each segment is subtracted from each of the bins it covers, and several estimators of spread are calculated from the residual values. The output text or table shows for each sample:

- Total number of segments (in the .cns file) – a large number of segments can indicate that the sample has either many real CNAs, or noisy coverage and therefore many spurious segments.
- Uncorrected sample standard deviation – this measure is prone to being inflated by a few outliers, such as may occur in regions of poor coverage or if the targets used with CNVkit analysis did not exactly match the capture. (Also note that the log2 ratio data are not quite normally distributed.) However, if a sample's standard deviation is drastically higher than the other estimates shown by the `metrics` command, that helpfully indicates the sample has some outlier bins.
- Median absolute deviation (MAD) – very robust against outliers, but less statistically efficient.
- Interquartile range (IQR) – another robust measure that is easy to understand.
- Tukey's biweight midvariance – a robust and efficient measure of spread.

Note that many small segments will fit noisy data better, shrinking the residuals used to calculate the other estimates of spread, even if many of the segments are spurious. One possible heuristic for judging the overall noisiness of each sample in a table is to multiply the number of segments by the biweight midvariance – the value will tend to be higher for unreliable samples. Check questionable samples for poor coverage (using e.g. bedtools, chanjo, IGV or Picard CalculateHsMetrics).

Finally, visualizing a sample with CNVkit's *scatter* command will often make it apparent whether a sample or the copy ratios within a genomic region can be trusted.

2.3.5 segmetrics

Calculate summary statistics of the residual bin-level log2 ratio estimates from the segment means, similar to the existing *metrics* command, but for each segment individually.

Results are output in the same format as the CNVkit segmentation file (.cns), with the stat names and calculated values printed in the "gene" column.

```
cnvkit.py segmetrics Sample.cnr -s Sample.cns --iqr
cnvkit.py segmetrics -s Sample.cn{s,r} --ci --pi
```

Supported stats:

- As in *metrics*: standard deviation (`--std`), median absolute deviation (`--mad`), inter-quartile range (`--iqr`), Tukey’s biweight midvariance (`--bivar`)
- confidence interval (`--ci`), estimated by bootstrap (100 resamples)
- prediction interval (`--pi`), estimated by the range between the 2.5-97.5 percentiles of bin-level log2 ratio values within the segment.

2.4 Compatibility and other I/O

2.4.1 import-picard

Convert Picard CalculateHsMetrics per-target coverage files (.csv) to the CNVkit .cnn format.

2.4.2 import-seg

Convert a file in the SEG format (e.g. the output of standard CBS or the GenePattern server) into one or more CNVkit .cns files.

The chromosomes in a SEG file may have been converted from chromosome names to integer IDs. Options in `import-seg` can help recover the original names.

- To add a “chr” prefix, use “-p chr”.
- To convert chromosome indices 23, 24 and 25 to the names “X”, “Y” and “M” (a common convention), use “-c human”.
- To use an arbitrary mapping of indices to chromosome names, use a comma-separated “key:value” string. For example, the human convention would be: “-c 23:X,24:Y,25:M”.

2.4.3 export

Convert copy number ratio tables (.cnr files) to another format.

bed

The segmented output from multiple samples (`*.cns`) can be exported to BED format to support a variety of other uses, such as viewing in a genome browser. The log2 ratio value of each segment is converted and rounded to an integer value, as required by the BED format. To get accurate copy number values, see the *call* command.

```
# Estimate integer copy number of each segment
cnvkit.py call Sample.cns -y -o Sample.call.cns
# Show estimated integer copy number of all regions
cnvkit.py export bed Sample.call.cns --show-neutral -y -o Sample.bed
```

The same format can also specify CNV regions to the FreeBayes variant caller with FreeBayes’s `--cnv-map` option:

```
# Show only CNV regions
cnvkit.py export bed *.call.cns -o all-samples.cnv-map.bed
```

Copy-number-neutral regions are not shown in the output by default, but can be included with the `--show-neutral` option.

cdt, jtv

A collection of probe-level copy ratio files (`*.cnr`) can be exported to Java TreeView via the standard CDT format or a plain text table:

```
cnvkit.py export jtv *.cnr -o Samples-JTV.txt
cnvkit.py export cdt *.cnr -o Samples.cdt
```

seg

Similarly, the segmentation files for multiple samples (`*.cns`) can be exported to the standard SEG format to be loaded in the Integrative Genomic Viewer (IGV):

```
cnvkit.py export seg *.cns -o Samples.seg
```

nexus-basic

The format `nexus-basic` can be loaded directly by the commercial program Biodiscovery Nexus Copy Number, specifying the "basic" input format in that program.

2.5 Additional scripts

genome2access.py: Calculate the sequence-accessible coordinates in chromosomes from the given reference genome, treating long spans of 'N' characters as the inaccessible regions.

CNVkit will compute "antitarget" bins only within the accessible genomic regions specified in the "access" file produced by this script. If there are many small excluded/inaccessible regions in the genome, then small, less-reliable antitarget bins would be squeezed into the remaining accessible regions. The `-s` option tells the script to ignore short regions that would otherwise be excluded as inaccessible, allowing larger antitarget bins to overlap them.

Additional regions to exclude can also be given with the `-x` option. This option can be used more than once to exclude several BED files listing different sets of regions. For example, "excludable" regions of poor mappability have been precalculated by others and are available from the UCSC FTP Server (see here for hg19).

refFlat2bed.py Generate a BED file of the genes or exons in the reference genome given in UCSC refFlat.txt format. (Download the input file from UCSC Genome Bioinformatics).

This script can be used in case the original BED file of targeted intervals is unavailable. Subsequent steps of the pipeline will remove probes that did not receive sufficient coverage, including those exons or genes that were not targeted by the sequencing library. However, CNVkit will give much better results if the true targeted intervals can be provided.

CHAPTER 3

FAQ

3.1 File formats

We've tried to use standard file formats where possible in CNVkit. However, in a few cases we have needed to extend the standard BED format to accommodate additional information.

All of the non-standard file formats used by CNVkit are tab-separated plain text and can be loaded in a spreadsheet program, R or other statistical analysis software for manual analysis, if desired.

3.1.1 BED

See: http://genome.ucsc.edu/FAQ/FAQformat.html#format1

Note that BED genomic coordinates are 0-indexed, like C or Python code, but unlike the 1-indexed GATK/Picard interval list format. (The first nucleotide of a 1000-basepair sequence has position 0, the last nucleotide has position 999, and the entire region is indicated by the range 0-1000.)

3.1.2 Target and antitarget bin-level coverages (.cnn)

Coverage of binned regions is saved in a tabular format similar to BED but with additional columns. Each row in the file indicates an on-target or off-target (antitarget, background) bin. Genomic coordinates are 0-indexed, like BED.

Column names are shown as the first line of the file:

- Chromosome or reference sequence name (`chromosome`)
- Start position (`start`)
- End position (`end`)
- Gene name (`gene`)
- Log2 mean coverage depth (`log2`)

Essentially the same tabular file format is used for coverages (.cnn), ratios (.cnr) and segments (.cns) emitted by CNVkit.

3.1.3 Copy number reference profile (.cnn)

In addition to the columns present in the "target" and "antitarget" .cnn files, the reference .cnn file has the columns:

- GC content of the sequence region (`gc`)

- RepeatMasker-masked proportion of the sequence region (`rmask`)
- Statistical spread or dispersion (`spread`)

The log2 coverage depth is the weighted average of coverage depths, excluding extreme outliers, observed at the corresponding bin in each the sample .cnn files used to construct the reference. The spread is a similarly weighted estimate of the standard deviation of normalized coverages in the bin.

To manually review potentially problematic genes in the built reference, you can sort the file by the "spread" column; bins with higher values are the noisy ones.

It is important to keep the copy number reference file consistent for the duration of a project, reusing the same reference for bias correction of all tumor samples in a cohort. If your library preparation protocol changes, it's usually best to build a new reference file and use the new file to analyze the samples prepared under the new protocol.

3.1.4 Bin-level log2 ratios (.cnr)

In addition to the BED-like `chromosome`, `start`, `end` and `gene` columns present in .cnn files, the .cnr file has the columns:

- Log2 ratio (`log2`)
- Proportional weight to be used for segmentation (`weight`)

The weight value is the inverse of the variance (i.e. square of `spread` in the reference) of normalized log2 coverage values seen among all normal samples at that bin. This value is used to weight the bin log2 ratio values during segmentation.

Also, when a genomic region is plotted with CNVkit's "scatter" command, the size of the plotted datapoints is proportional to the weight of each point used in segmentation – a relatively small point indicates a less reliable bin.

3.1.5 Segmented log2 ratios (.cns)

In addition to the `chromosome`, `start`, `end`, `gene` and `log2` columns present in .cnr files, the .cns file format has the additional column `probes`, indicating the number of bins covered by the segment.

The `gene` column does not contain actual gene names. Rather, the sign of the segment's copy ratio value is indicated by 'G' if greater than zero or 'L' if less than zero.

3.2 Calling gains and losses

The relationship between the observed copy ratio and the true underlying copy number depends on tumor cell fraction (purity), genome ploidy (which may be heterogeneous in a tissue sample), and the size of the subclonal population containing the CNA. Because of these ambiguities, CNVkit only reports the estimated log2 copy ratio, and does not currently attempt a formal statistical test for estimating integer copy number.

In a diploid genome, a single-copy gain in a perfectly pure, homogeneous sample has a copy ratio of 3/2. In log2 scale, this is log2(3/2) = 0.585, and a single-copy loss is log2(1/2) = -1.0.

In the *diagram* plot, for the sake of providing a clean visualization of confidently called CNAs, the default threshold to label genes is 0.5. This threshold will tend to display gene amplifications, fully clonal single-copy gains in fairly pure samples, most single-copy losses, and complete deletions.

When using the *gainloss* command, choose a threshold to suit your needs depending on your knowledge of the sample's purity, heterogeneity, and likely features of interest. As a starting point, try 0.1 or 0.2 if you are going to do your own filtering downstream, or 0.3 if not.

The *call* command implements two simple methods to convert the log2 ratios in a segmented .cns file to absolute integer copy number values. The *segmetrics* command computes segment-level summary statistics that can be used to evaluate the reliability of each segment. Future releases of CNVkit will integrate further statistical testing to help make meaningful variant calls from the log2 copy ratio data.

3.3 Tumor heterogeneity

DNA samples extracted from solid tumors are rarely completely pure. Stromal or other normal cells and distinct subclonal tumor-cell populations are typically present in a sample, and can confound attempts to fit segmented log2 ratio values to absolute integer copy numbers.

CNVkit provides several points of integration with existing tools and methods for dealing with tumor heterogeneity and normal-cell contamination.

3.3.1 Inferring tumor purity and subclonal population fractions

The third-party program THetA2 can be used to estimate tumor cell content and infer integer copy number of tumor subclones in a sample. CNVkit provides wrappers for exporting segments to THetA2's input format and importing THetA2's result file as CNVkit's segmented .cns files.

Using CNVkit with THetA2

THetA2's input file is a BED-like file, typically with the extension `.input`, listing the read counts within each copy-number segment in a pair of tumor and normal samples. CNVkit can generate this file given the CNVkit-inferred tumor segmentation (.cns) and normal copy log2-ratios (.cnr) or copy number reference file (.cnn). This bypasses the initial step of THetA2, CreateExomeInput, which counts the reads in each sample's BAM file.

After running the CNVkit Copy number calling pipeline on a sample, create the THetA2 input file:

```
# From a paired normal sample
cnvkit.py export theta Sample_Tumor.cns Sample_Normal.cnr -o Sample.theta2.input
# From an existing CNVkit reference
cnvkit.py export theta Sample_Tumor.cns reference.cnn -o Sample.theta2.input
```

Then, run THetA2 (assuming the program was unpacked at `/path/to/theta2/`):

```
# Generates Sample.theta2.BEST.results:
/path/to/theta2/bin/RunTHetA Sample.theta2.input
# Parameters for low-quality samples:
/path/to/theta2/python/RunTHetA.py Sample.theta2.input -n 2 -k 4 -m .90 --FORCE --NUM_PROCESSES `npro
```

Finally, import THetA2's results back into CNVkit's .cns format, matching the original segmentation (.cns) to the THetA2-inferred absolute copy number values.:

```
cnvkit.py import-theta Sample_Tumor.cns Sample.theta2.BEST.results
```

THetA2 adjusts the segment log2 values to the inferred cellularity of each detected subclone; this can result in one or two .cns files representing subclones if more than one clonal tumor cell population was detected. THetA2 also performs some significance testing of each segment representing a CNA, so there may be fewer segments derived from THetA2 than were originally found by CNVkit.

The segment values are still log2-transformed in the resulting .cns files, for convenience in plotting etc. with CNVkit. These files are also easily converted to other formats using the *export* command.

3.3.2 Adjusting copy ratios and segments for normal cell contamination

CNVkit's *call* command converts the segmented log2 ratio estimates to absolute integer copy numbers.

The `clonal` method in this command uses an estimate of tumor fraction (from any source) to directly rescale segment log2 ratio values and round them to the nearest integer copy number. Example with tumor purity of 60% and a male reference:

```
cnvkit.py call -m clonal Sample.cns --purity 0.6 -y -o Sample.call-clonal.cns
```

Alternatively, if the tumor cell fraction is not known, hard thresholds can be used instead:

> cnvkit.py call -m threshold Sample.cns -y -o Sample.call-threshold.cns

3.3.3 Export integer copy numbers as BED

The *export* `bed` command emits integer copy number calls in standard BED format:

```
cnvkit.py export bed Sample.cns -y -o Sample.bed
```

3.4 Whole-genome sequencing and targeted amplicon capture

CNVkit is designed for use on **hybrid capture** sequencing data, where off-target reads are present and can be used improve copy number estimates.

If necessary, CNVkit can be used on **whole-genome sequencing** (WGS) datasets by specifying the genome's sequencing-accessible regions as the "targets", avoiding "antitargets", and using a gene annotation database to label genes in the resulting BED file:

```
cnvkit.py batch ... -t data/access-5kb-mappable.hg19.bed -g data/access-5kb-mappable.hg19.bed --split
```

Or:

```
cnvkit.py target data/access-5kb-mappable.hg19.bed --split --annotate refFlat.txt -o Targets.bed
cnvkit.py antitarget data/access-5kb-mappable.hg19.bed -g data/access-5kb-mappable.hg19.bed -o Backgr
```

This produces a "target" binning of the entire sequencing-accessible area of the genome, and empty "antitarget" files which CNVkit will handle safely from version 0.3.4 onward.

Similarly, to use CNVkit on **targeted amplicon sequencing** data instead – although this is not recommended – you can exclude all off-target regions from the analysis by passing the target BED file as the "access" file as well:

```
cnvkit.py batch ... -t Targeted.bed -g Targeted.bed ...
```

Or:

```
cnvkit.py antitarget Targeted.bed -g Targeted.bed -o Background.bed
```

However, this approach does not collect any copy number information between targeted regions, so it should only be used if you have in fact prepared your samples with a targeted amplicon sequencing protocol. It also does not attempt to normalize each amplicon at the gene level, though this may be addressed in a future version of CNVkit.

CHAPTER 4

Python API

4.1 Python API (cnvlib package)

4.1.1 Module `cnvlib` contents

`cnvlib.`**`read`**(*fname*)

Parse a file as a copy number or copy ratio table (.cnn, .cnr).

4.1.2 Submodules

`cnarray`

The core object used throughout CNVkit. For your own scripting, you can usually accomplish what you need using CopyNumArray methods. Definitions for the core data structure, a copy number array.

class `cnvlib.cnarray.`**`CopyNumArray`**(*sample_id*, *extra_column_names=()*)

Bases: `object`

An array of genomic intervals, treated like aCGH probes.

`add_columns`(***columns*)

Create a new CNA, adding the specified extra columns to this CNA.

`by_bin`(*bins*)

Group rows by another CopyNumArray; trim row start/end to bin edges.

Returns an iterable of (bin, CopyNumArray of overlapping cnarray rows))

If a probe overlaps with a bin boundary, the probe start or end position is replaced with the bin boundary position. Probes outside any segments are skipped. This is appropriate for most other comparisons between CopyNumArray objects.

`by_chromosome`()

Iterate over probes grouped by chromosome name.

`by_gene`(*ignore=('-', 'CGH', '.')*)

Iterate over probes grouped by gene name.

Emits pairs of (gene name, CNA of rows with same name)

Groups each series of intergenic bins as a 'Background' gene; any 'Background' bins within a gene are grouped with that gene. Bins with names in *ignore* are treated as 'Background' bins, but retain their name.

by_segment(*segments*)

Group cnarray rows by the segments that row midpoints land in.

Returns an iterable of segments and rows grouped by overlap with each segment.

Note that segments don't necessarily cover all probes (some near telo/centromeres may have been dropped as outliers during segmentation). These probes are grouped with the nearest segment, so the endpoint of the first/last probe may not match the corresponding segment endpoint. This is appropriate if the segments were obtained from this probe array.

center_all(*peak=False*)

Recenter coverage values to the autosomes' average (in-place).

chromosome

copy()

Create an independent copy of this object.

coverage

drop_extra_columns()

Remove any optional columns from this CopyNumArray.

Returns a new copy with only the core columns retained: log2 value, chromosome, start, end, bin name.

end

classmethod from_array(*sample_id*, *array*)

classmethod from_columns(*sample_id*, ***columns*)

classmethod from_rows(*sample_id*, *row_data*, *extra_keys=()*)

gene

in_range(*chrom*, *start=0*, *end=None*, *trim=False*)

Get the CopyNumArray portion within the given genomic range.

If trim=True, include bins straddling the range boundaries, and trim the bins endpoints to the boundaries.

labels()

merge(*other*)

Combine this array's data with another CopyNumArray (in-place).

Any optional columns must match between both arrays.

classmethod read(*infile*, *sample_id=None*)

Parse a tabular table of coverage data from a handle or filename.

select(*selector=None*, ***kwargs*)

Take a subset of rows where the given condition is true.

Arguments can be a function (lambda expression) returning a bool, which will be used to select True rows, and/or keyword arguments like gene="Background" or chromosome="chr7", which will select rows where the keyed field equals the specified value.

shuffle()

Randomize the order of bins in this array (in-place).

sort(*key=None*)

Sort the bins in this array (in-place).

Optional argument 'key' is one of:

- a function that computes a sorting key from a CopyNumArray row
- a string identifier for an existing data column
- a list/array/iterable of precomputed keys equal in length to the number of rows in this CopyNumArray.

By default, bins are sorted by chromosomal coordinates.

squash_genes(*ignore=('-', 'CGH', '.')*, *squash_background=False*, *summary_stat=<function biweight_location>*)

Combine consecutive bins with the same targeted gene name.

The *ignore* parameter lists bin names that not be counted as genes to be output.

Parameter *summary_stat* is a function that summarizes an array of coverage values to produce the "squashed" gene's coverage value. By default this is the biweight location, but you might want median, mean, max, min or something else in some cases.

Optional columns, if present, are dropped.

start

to_array(*array*)

to_rows(*rows*)

Like from_rows, reusing this instance's metadata.

write(*outfile=<open file '<stdout>', mode 'w'>*)

Write coverage data to a file or handle in tabular format.

This is similar to BED or BedGraph format, but with extra columns.

To combine multiple samples in one file and/or convert to another format, see the 'export' subcommand.

cnvlib.cnarray.**row2label**(*row*)

commands

The public API for each of the commands defined in the CNVkit workflow. Command-line interface and corresponding API for CNVkit.

class cnvlib.commands.**SerialPool**

Bases: object

Mimic the multiprocessing.Pool interface, but run in serial.

apply_async(*func*, *args*)

Just call the function.

close()

join()

cnvlib.commands.**batch_make_reference**(*normal_bams*, *target_bed*, *antitarget_bed*, *male_reference*, *fasta*, *annotate*, *short_names*, *split*, *target_avg_size*, *access*, *antitarget_avg_size*, *antitarget_min_size*, *output_reference*, *output_dir*, *processes*, *by_count*)

Build the CN reference from normal samples, targets and antitargets.

cnvlib.commands.**batch_run_sample**(*bam_fname*, *target_bed*, *antitarget_bed*, *ref_fname*, *output_dir*, *male_reference=False*, *scatter=False*, *diagram=False*, *rlibpath=None*, *by_count=False*)

Run the pipeline on one BAM file.

`cnvlib.commands.batch_write_coverage(bed_fname, bam_fname, out_fname, by_count)`

Run coverage on one sample, write to file.

`cnvlib.commands.create_heatmap(filenames, show_chromosome=None, do_desaturate=False)`

Plot copy number for multiple samples as a heatmap.

`cnvlib.commands.create_loh(variants, min_depth=20, do_trend=False, sample_id=None)`

Plot allelic frequencies at each variant position in a VCF file.

`cnvlib.commands.csvstring(text)`

`cnvlib.commands.do_antitarget(target_bed, access_bed=None, avg_bin_size=150000, min_bin_size=None)`

Derive a background/antitarget BED file from a target BED file.

`cnvlib.commands.do_breaks(probes, segments, min_probes=1)`

List the targeted genes in which a copy number breakpoint occurs.

`cnvlib.commands.do_call(segments, method, ploidy=2, purity=None, is_reference_male=False, is_sample_female=False, thresholds=(-1.1, -0.3, 0.2, 0.7))`

`cnvlib.commands.do_coverage(bed_fname, bam_fname, by_count=False)`

Calculate coverage in the given regions from BAM read depths.

`cnvlib.commands.do_fix(target_raw, antitarget_raw, reference, do_gc=True, do_edge=True, do_rmask=True)`

Combine target and antitarget coverages and correct for biases.

`cnvlib.commands.do_gainloss(probes, segments=None, male_reference=False, threshold=0.2, min_probes=3)`

Identify targeted genes with copy number gain or loss.

`cnvlib.commands.do_reference(target_fnames, antitarget_fnames, fa_fname=None, male_reference=False)`

Compile a coverage reference from the given files (normal samples).

`cnvlib.commands.do_reference_flat(target_list, antitarget_list, fa_fname=None, male_reference=False)`

Compile a neutral-coverage reference from the given intervals.

Combines the intervals, shifts chrX values if requested, and calculates GC and RepeatMasker content from the genome FASTA sequence.

`cnvlib.commands.do_scatter(pset_cvg, pset_seg=None, vcf_fname=None, show_chromosome=None, show_gene=None, show_range=None, background_marker=None, do_trend=False, window_width=1000000.0, sample_id=None, min_variant_depth=20, y_min=None, y_max=None)`

Plot probe log2 coverages and CBS calls together.

`cnvlib.commands.do_targets(bed_fname, out_fname, annotate=None, do_short_names=False, do_split=False, avg_size=266.6666666666667)`

Transform bait intervals into targets more suitable for CNVkit.

`cnvlib.commands.parse_args(args=None)`

Parse the command line.

`cnvlib.commands.pick_pool(nprocs)`

`cnvlib.commands.print_version(_args)`

Display this program’s version.

antitarget

Supporting functions for the ‘antitarget’ command.

`cnvlib.antitarget.`**`find_background_regions`**(*access_chroms*, *target_chroms*, *pad_size*)

Take coordinates of accessible regions and targets; emit antitargets.

Note that a chromosome must be present in the target library in order to be included in the antitargets generated here. So, if chrY is missing from your output files, it’s probably because it had no targets.

`cnvlib.antitarget.`**`get_background`**(*target_bed*, *access_bed*, *avg_bin_size*, *min_bin_size*)

Generate background intervals from target intervals.

Procedure:

- Invert target intervals
- Subtract the inverted targets from accessible regions
- For each of the resulting regions:
  - Shrink by a fixed margin on each end
  - If it’s smaller than min_bin_size, skip
  - Divide into equal-size (region_size/avg_bin_size) portions
  - Emit the (chrom, start, end) coords of each portion

`cnvlib.antitarget.`**`group_coords`**(*coordinates*)

Group chromosomal coordinates into a dictionary.

`cnvlib.antitarget.`**`guess_chromosome_regions`**(*target_chroms*, *telomere_size*)

Determine (minimum) chromosome lengths from target coordinates.

core

CNV utilities.

`cnvlib.core.`**`assert_equal`**(*msg*, ***values*)

Evaluate and compare two or more values for equality.

Sugar for a common assertion pattern. Saves re-evaluating (and retyping) the same values for comparison and error reporting.

Example:

```
>>> assert_equal("Mismatch", expected=1, saw=len(['xx', 'yy']))
...
ValueError: Mismatch: expected = 1, saw = 2
```

`cnvlib.core.`**`check_unique`**(*items*, *title*)

Ensure all items in an iterable are identical; return that one item.

`cnvlib.core.`**`expect_flat_cvg`**(*cnarr*, *is_male_reference=None*, *chr_x=None*)

Get the uninformed expected copy ratios of each bin.

Create an array of log2 coverages like a “flat” reference.

This is a neutral copy ratio at each autosome (log2 = 0.0) and sex chromosomes based on whether the reference is male (XX or XY).

`cnvlib.core.`**`fbase`**(*fname*)

Strip directory and all extensions from a filename.

cnvlib.core.**get_relative_chrx_cvg**(*probes*, *chr_x=None*)

Get the relative log-coverage of chrX in a sample.

cnvlib.core.**guess_chr_x**(*probes*)

cnvlib.core.**guess_xx**(*probes*, *male_reference=False*, *chr_x=None*, *verbose=True*)

Guess whether a sample is female from chrX relative coverages.

Recommended cutoff values: -0.5 – raw target data, not yet corrected +0.5 – probe data already corrected on a male profile

cnvlib.core.**parse_tsv**(*infile*, *keep_header=False*)

Parse a tabular data table into an iterable of lists.

Rows are split on tabs. Header row is optionally included in the output.

cnvlib.core.**rbase**(*fname*)

Strip directory and final extension from a filename.

cnvlib.core.**shift_xx**(*probes*, *male_reference=False*, *chr_x=None*)

Adjust chrX coverages (divide in half) for apparent female samples.

cnvlib.core.**sorter_chrom**(*label*)

Create a sorting key from chromosome label.

Sort by integers first, then letters or strings. The prefix "chr" (case-insensitive), if present, is stripped automatically for sorting.

E.g. chr1 < chr2 < chr10 < chrX < chrY

cnvlib.core.**sorter_chrom_at**(*index*)

Create a sort key function that gets chromosome label at a list index.

cnvlib.core.**write_tsv**(*outfname*, *table*, *colnames=None*)

Write the CGH file.

coverage

Supporting functions for the 'antitarget' command.

cnvlib.coverage.**bam_total_reads**(*bam_fname*)

Count the total number of mapped reads in a BAM file.

Uses the BAM index to do this quickly.

cnvlib.coverage.**bedcov**(*bed_fname*, *bam_fname*)

Calculate depth of all regions in a BED file via samtools (pysam) bedcov.

i.e. mean pileup depth across each region.

cnvlib.coverage.**filter_column**(*col*)

Count the number of filtered reads in a pileup column.

cnvlib.coverage.**filter_read**(*read*)

True if the given read should be counted towards coverage.

cnvlib.coverage.**interval_coverages**(*bed_fname*, *bam_fname*, *by_count*)

Calculate log2 coverages in the BAM file at each interval.

cnvlib.coverage.**interval_coverages_count**(*bed_fname*, *bam_fname*)

Calculate log2 coverages in the BAM file at each interval.

cnvlib.coverage.**interval_coverages_pileup**(*bed_fname*, *bam_fname*)

Calculate log2 coverages in the BAM file at each interval.

`cnvlib.coverage.`**`region_depth_count`**(*bamfile*, *chrom*, *start*, *end*)

Calculate depth of a region via pysam count.

i.e. counting the number of read starts in a region, then scaling for read length and region width to estimate depth.

Coordinates are 0-based, per pysam.

diagram

Chromosome diagram drawing functions.

This uses and abuses Biopython's BasicChromosome module. It depends on ReportLab, too, so we isolate this functionality here so that the rest of CNVkit will run without it. (And also to keep the codebase tidy.)

`cnvlib.diagram.`**`bc_chromosome_draw_label`**(*self*, *cur_drawing*, *label_name*)

Monkeypatch to Bio.Graphics.BasicChromosome.Chromosome._draw_label.

Draw a label for the chromosome. Mod: above the chromosome, not below.

`cnvlib.diagram.`**`bc_organism_draw`**(*org*, *title*, *wrap=12*)

Modified copy of Bio.Graphics.BasicChromosome.Organism.draw.

Instead of stacking chromosomes horizontally (along the x-axis), stack rows vertically, then proceed with the chromosomes within each row.

Arguments:

- title: The output title of the produced document.

`cnvlib.diagram.`**`build_chrom_diagram`**(*features*, *chr_sizes*, *sample_id*)

Create a PDF of color-coded features on chromosomes.

`cnvlib.diagram.`**`create_diagram`**(*cnarr*, *segarr*, *threshold*, *min_probes*, *outfname*, *male_reference*)

Create the diagram.

export

Export CNVkit objects and files to other formats.

class `cnvlib.export.`**`ProbeInfo`**(*label*, *chrom*, *start*, *end*, *gene*)

Bases: `tuple`

`chrom`

Alias for field number 1

`end`

Alias for field number 3

`gene`

Alias for field number 4

`label`

Alias for field number 0

`start`

Alias for field number 2

`cnvlib.export.`**`calculate_theta_fields`**(*seg*, *ref_rows*, *chrom_id*)

Convert a segment's info to a row of THetA input.

For the normal/reference bin count, take the mean of the bin values within each segment so that segments match between tumor and normal.

`cnvlib.export.`**`create_chrom_ids`**`(`*segments*`)`
Map chromosome names to integers in the order encountered.

`cnvlib.export.`**`export_bed`**`(`*sample_fnames*, *args*`)`
Export to BED format.

For each region in each sample which does not have neutral copy number (equal to 2 or the value set by –ploidy), the columns are:

- reference sequence name
- start (0-indexed)
- end
- sample name or given label
- integer copy number

`cnvlib.export.`**`export_freebayes`**`(`*sample_fnames*, *args*`)`
Export to FreeBayes –cnv-map format.

Which is BED-like, for each region in each sample which does not have neutral copy number (equal to 2 or the value set by –ploidy), with columns:

- reference sequence
- start (0-indexed)
- end
- sample name
- copy number

`cnvlib.export.`**`export_nexus_basic`**`(`*sample_fname*`)`
Biodiscovery Nexus Copy Number “basic” format.

Only represents one sample per file.

`cnvlib.export.`**`export_seg`**`(`*sample_fnames*`)`
SEG format for copy number segments.

Segment breakpoints are not the same across samples, so samples are listed in serial with the sample ID as the left column.

`cnvlib.export.`**`export_theta`**`(`*tumor*, *reference*`)`
Convert tumor segments and normal .cnr or reference .cnn to THetA input.

Follows the THetA segmentation import script but avoid repeating the pileups, since we already have the mean depth of coverage in each target bin.

The options for average depth of coverage and read length do not matter crucially for proper operation of THetA; increased read counts per bin simply increase the confidence of THetA’s results.

THetA2 input format is tabular, with columns: ID, chrm, start, end, tumorCount, normalCount

where chromosome IDs (“chrm”) are integers 1 through 24.

`cnvlib.export.`**`fmt_cdt`**`(`*sample_ids*, *rows*`)`
Format as CDT.

`cnvlib.export.`**`fmt_gct`**`(`*sample_ids*, *rows*`)` cnvlib.export.**fmt_jtv**(*sample_ids*, *rows*)

Format for Java TreeView.

cnvlib.export.**fmt_multi**(*sample_ids*, *rows*)

cnvlib.export.**fmt_vcf**(*sample_ids*, *rows*)

cnvlib.export.**merge_rows**(*rows*)

Combine equivalent rows of coverage data across multiple samples.

Check that probe info matches across all samples, then merge the log2 coverage values.

Input: a list of individual rows corresponding to the same probes from different coverage files. Output: a list starting with the single common Probe object, followed by the log2 coverage values from each sample, in order.

cnvlib.export.**merge_samples**(*filenames*)

Merge probe values from multiple samples into a 2D table (of sorts).

Input: dict of {sample ID: (probes, values)}

Output: list-of-tuples: (probe, log2 coverages...)

cnvlib.export.**row_to_probe_coverage**(*row*)

Repack a parsed row into a ProbeInfo instance and coverage value.

cnvlib.export.**segments2bed**(*segments*, *label*, *ploidy*, *is_reference_male*, *show_neutral*)

Convert a copy number array to a BED-like format.

cnvlib.export.**segments2freebayes**(*segments*, *sample_name*, *ploidy*, *purity*, *is_reference_male*, *is_sample_female*)

Convert a copy number array to a BED-like format.

fix

Supporting functions for the 'fix' command.

cnvlib.fix.**apply_weights**(*cnarr*, *ref_matched*, *epsilon=0.0001*)

Calculate weights for each bin.

Weights are derived from:

- bin sizes
- average bin coverage depths in the reference
- the "spread" column of the reference.

cnvlib.fix.**center_by_window**(*pset*, *fraction*, *sort_key*)

Smooth out biases according to the trait specified by sort_key.

E.g. correct GC-biased probes by windowed averaging across similar-GC probes; or for similar interval sizes.

cnvlib.fix.**edge_gain**(*target_size*, *insert_size*, *gap_size*)

Calculate coverage gain from a neighboring bait's flanking reads.

Letting i = insert size, t = target size, g = gap to neighboring bait, the gain of coverage due to a nearby bait, if g < i, is:

```
    (i-g)^2 / 4it
```

If the neighbor flank extends beyond the target (t+g < i), reduce by:

```
    (i-t-g)^2 / 4it
```

cnvlib.fix.**edge_loss**(*target_size*, *insert_size*)

Calculate coverage loss at the edges of a baited region.

Letting i = insert size and t = target size, the proportional loss of coverage near the two edges of the baited region (combined) is:

```
i/2t
```

If the "shoulders" extend outside the bait $(t < i), reduce by:

```
(i-t)^2 / 4it
```

on each side, or (i-t)^2 / 2it total.

cnvlib.fix.**load_adjust_coverages**(*pset*, *ref_pset*, *fix_gc*, *fix_edge*, *fix_rmask*)

Load and filter probe coverages; correct using reference and GC.

cnvlib.fix.**make_edge_sorter**(*target_probes*, *margin*)

Create a sort-key function for tiling edge effects.

cnvlib.fix.**match_ref_to_probes**(*ref_pset*, *probes*)

Filter the reference probes to match the target or antitarget probe set.

importers

Import from other formats to the CNVkit format.

cnvlib.importers.**find_picard_files**(*file_and_dir_names*)

Search the given paths for 'targetcoverage' CSV files.

Per the convention we use in our Picard applets, the target coverage file names end with '.targetcoverage.csv'; anti-target coverages end with '.antitargetcoverage.csv'.

cnvlib.importers.**import_seg**(*segfname*, *chrom_names*, *chrom_prefix*, *from_log10*)

Parse a SEG file. Emit pairs of (sample ID, CopyNumArray)

Values are converted from log10 to log2.

***chrom_names*:** Map (string) chromosome IDs to names. (Applied before chrom_prefix.) e.g. {'23': 'X', '24': 'Y', '25': 'M'}

***chrom_prefix*: prepend this string to chromosome names** (usually 'chr' or None)

cnvlib.importers.**load_targetcoverage_csv**(*fname*)

Parse a target or antitarget coverage file (.csv) into a CopyNumArray.

These files are generated by Picard CalculateHsMetrics. The fields of the .csv files are actually separated by tabs, not commas.

CSV column names: chrom (str), start, end, length (int), name (str), %gc, mean_coverage, normalized_coverage (float)

cnvlib.importers.**parse_theta_results**(*fname*)

Parse THetA results into a data structure.

Columns: NLL, mu, C, p*

cnvlib.importers.**unpipe_name**(*name*)

Fix the duplicated gene names Picard spits out.

Return a string containing the single gene name, sans duplications and pipe characters.

Picard CalculateHsMetrics combines the labels of overlapping intervals by joining all labels with '|', e.g. 'BRAF|BRAF' – no two distinct targeted genes actually overlap, though, so these dupes are redundant.

Also, in our convention, 'CGH' probes are selected intergenic regions, not meaningful gene names, so 'CGH|FOO' resolves as 'FOO'.

metrics

Robust estimators of central tendency and scale.

For use in evaluating performance of copy number estimation.

See: http://en.wikipedia.org/wiki/Robust_measures_of_scale http://astropy.readthedocs.org/en/latest/_modules/astropy/stats/funcs.html

`cnvlib.metrics.`**`biweight_location`**(*a*, *initial=None*, *c=6.0*, *epsilon=0.0001*)
Compute the biweight location for an array.

The biweight is a robust statistic for determining the central location of a distribution.

`cnvlib.metrics.`**`biweight_midvariance`**(*a*, *initial=None*, *c=9.0*, *epsilon=0.0001*)
Compute the biweight midvariance for an array.

The biweight midvariance is a robust statistic for determining the midvariance (i.e. the standard deviation) of a distribution.

See: http://en.wikipedia.org/wiki/Robust_measures_of_scale#The_biweight_midvariance http://astropy.readthedocs.org/en/latest/_modules/astropy/stats/funcs.html

`cnvlib.metrics.`**`ests_of_scale`**(*deviations*)
Estimators of scale: standard deviation, MAD, biweight midvariance.

Calculates all of these values for an array of deviations and returns them as a tuple.

`cnvlib.metrics.`**`interquartile_range`**(*a*)
Compute the difference between the array's first and third quartiles.

`cnvlib.metrics.`**`median_absolute_deviation`**(*a*, *scale_to_sd=True*)
Compute the median absolute deviation (MAD) of array elements.

The MAD is defined as: `median(abs(a - median(a)))`.

See: http://en.wikipedia.org/wiki/Median_absolute_deviation

`cnvlib.metrics.`**`probe_deviations_from_segments`**(*probes*, *segments*, *skip_low=True*)
Difference in CN estimate of each probe from its segment.

`cnvlib.metrics.`**`q_n`**(*a*)
Rousseeuw & Croux's (1993) Q_n, an alternative to MAD.

`Qn := Cn first quartile of (|x_i - x_j|:  i < j)`

where Cn is a constant depending on n.

Finite-sample correction factors must be used to calibrate the scale of Qn for small-to-medium-sized samples.

> n E[Qn] – —– 10 1.392 20 1.193 40 1.093 60 1.064 80 1.048 100 1.038 200 1.019

`cnvlib.metrics.`**`segment_mean`**(*cnarr*)
Weighted average of bin log2 values, ignoring too-low-coverage bins.

ngfrills

NGS utilities.

cnvlib.ngfrills.**call_quiet**(**args*)

Safely run a command and get stdout; print stderr if there's an error.

Like subprocess.check_output, but silent in the normal case where the command logs unimportant stuff to stderr. If there is an error, then the full error message(s) is shown in the exception message.

cnvlib.ngfrills.**ensure_path**(*fname*)

Create dirs and move an existing file to avoid overwriting, if necessary.

If a file already exists at the given path, it is renamed with an integer suffix to clear the way.

cnvlib.ngfrills.**safe_write**(**args*, ***kwds*)

Write to a filename or file-like object with error handling.

If given a file name, open it. If the path includes directories that don't exist yet, create them. If given a file-like object, just pass it through.

cnvlib.ngfrills.**temp_write_text**(**args*, ***kwds*)

Save text to a temporary file.

NB: This won't work on Windows b/c the file stays open.

params

Hard-coded parameters for CNVkit. These should not change between runs.

plots

Plotting utilities.

cnvlib.plots.**chromosome_sizes**(*probes*, *to_mb=False*)

Create an ordered mapping of chromosome names to sizes.

cnvlib.plots.**cvg2rgb**(*cvg*, *desaturate*)

Choose a shade of red or blue representing log2-coverage value.

cnvlib.plots.**gene_coords_by_name**(*probes*, *names*)

Find the chromosomal position of each named gene in probes.

Returns a dict: {chromosome: [(start, end, gene name), ...]}

cnvlib.plots.**gene_coords_by_range**(*probes*, *chrom*, *start*, *end*, *skip=('Background', 'CGH', '-', '.')*)

Find the chromosomal position of all genes in a range.

Returns a dict: {chromosome: [(start, end, gene), ...]}

cnvlib.plots.**group_snvs_by_segments**(*snv_posns*, *snv_freqs*, *segments*, *chrom*)

Group SNP allele frequencies by segment.

Return an iterable of: start, end, value.

cnvlib.plots.**limit**(*x*, *lower*, *upper*)

Limit x to between lower and upper bounds.

cnvlib.plots.**parse_range**(*text*)

Parse a chromosomal range specification.

Range spec string should look like: 'chr1:1234-5678'

cnvlib.plots.**partition_by_chrom**(*chrom_snvs*)

Group the tumor shift values by chromosome (for statistical testing).

`cnvlib.plots.`**`plot_chromosome`**(*axis*, *probes*, *segments*, *chromosome*, *sample*, *genes*, *background_marker=None*, *do_trend=False*, *y_min=None*, *y_max=None*)

Draw a scatter plot of probe values with CBS calls overlaid.

Argument 'genes' is a list of tuples: (start, end, gene name)

`cnvlib.plots.`**`plot_genome`**(*axis*, *probes*, *segments*, *pad*, *do_trend=False*, *y_min=None*, *y_max=None*)

Plot coverages and CBS calls for all chromosomes on one plot.

`cnvlib.plots.`**`plot_loh`**(*axis*, *chrom_snvs*, *chrom_sizes*, *segments*, *do_trend*, *pad*)

Plot a scatter-plot of SNP chromosomal positions and shifts.

`cnvlib.plots.`**`plot_x_dividers`**(*axis*, *chromosome_sizes*, *pad*)

Plot vertical dividers and x-axis labels given the chromosome sizes.

Returns a table of the x-position offsets of each chromosome.

Draws vertical black lines between each chromosome, with padding. Labels each chromosome range with the chromosome name, centered in the region, under a tick. Sets the x-axis limits to the covered range.

`cnvlib.plots.`**`probe_center`**(*row*)

Return the midpoint of the probe location.

`cnvlib.plots.`**`test_loh`**(*bins*, *alpha=0.0025*)

Test each chromosome's SNP shifts and the combined others'.

The statistical test is Mann-Whitney, a one-sided non-parametric test for difference in means.

reference

Supporting functions for the 'reference' command.

`cnvlib.reference.`**`bed2probes`**(*bed_fname*)

Create neutral-coverage probes from intervals.

`cnvlib.reference.`**`calculate_gc_lo`**(*subseq*)

Calculate the GC and lowercase (RepeatMasked) content of a string.

`cnvlib.reference.`**`combine_probes`**(*filenames*, *fa_fname*, *is_male_reference*)

Calculate the median coverage of each bin across multiple samples.

Input: List of .cnn files, as generated by 'coverage' or 'import-picard'. *fa_fname*: fil columns for GC and RepeatMasker genomic values.

Returns: A single CopyNumArray summarizing the coverages of the input samples, including each bin's "average" coverage, "spread" of coverages, and genomic GC content.

`cnvlib.reference.`**`get_fasta_stats`**(*probes*, *fa_fname*)

Calculate GC and RepeatMasker content of each bin in the FASTA genome.

`cnvlib.reference.`**`mask_bad_probes`**(*probes*)

Flag the probes with excessively low or inconsistent coverage.

Returns a bool array where True indicates probes that failed the checks.

`cnvlib.reference.`**`reference2regions`**(*reference*, *coord_only=False*)

Extract iterables of target and antitarget regions from a reference CNA.

Like loading two BED files with ngfrills.parse_regions.

cnvlib.reference.**warn_bad_probes**(*probes*)

Warn about target probes where coverage is poor.

Prints a formatted table to stderr.

reports

Supporting functions for the text/tabular-reporting commands.

Namely: breaks, gainloss.

cnvlib.reports.**gainloss_by_gene**(*probes*, *threshold*)

Identify genes where average bin copy ratio value exceeds *threshold*.

NB: Must shift sex-chromosome values beforehand with core.shift_xx, otherwise all chrX/chrY genes may be reported gained/lost.

cnvlib.reports.**gainloss_by_segment**(*probes*, *segments*, *threshold*)

Identify genes where segmented copy ratio exceeds *threshold*.

NB: Must shift sex-chromosome values beforehand with core.shift_xx, otherwise all chrX/chrY genes may be reported gained/lost.

cnvlib.reports.**get_breakpoints**(*intervals*, *segments*, *min_probes*)

Identify CBS segment breaks within the targeted intervals.

cnvlib.reports.**get_gene_intervals**(*all_probes*, *skip=('Background', 'CGH', '-', '.')*)

Tally genomic locations of each targeted gene.

Return a dict of chromosomes to a list of tuples: (gene name, start, end).

cnvlib.reports.**group_by_genes**(*probes*)

Group probe and coverage data by gene.

Return an iterable of genes, in chromosomal order, associated with their location and coverages:

> [(gene, chrom, start, end, [coverages]), ...]

segmentation

Segmentation of copy number values.

cnvlib.segmentation.**do_segmentation**(*probes_fname*, *save_dataframe*, *method*, *rlibpath=None*)

Infer copy number segments from the given coverage table.

cnvlib.segmentation.**repair_segments**(*segments*, *orig_probes*)

Post-process segmentation output.

1. Ensure every chromosome has at least one segment.
2. Ensure first and last segment ends match 1st/last bin ends (but keep log2 as-is).

cnvlib.segmentation.**squash_segments**(*seg_pset*)

Combine contiguous segments.

smoothing

Signal smoothing functions.

cnvlib.smoothing.**check_inputs**(*x*, *width*)

Transform width into a half-window size.

width is either a fraction of the length of *x* or an integer size of the whole window. The output half-window size is truncated to the length of *x* if needed.

cnvlib.smoothing.**fit_edges**(*x*, *y*, *wing*, *polyorder=3*)

Apply polynomial interpolation to the edges of y, in-place.

Calculates a polynomial fit (of order *polyorder*) of *x* within a window of width twice *wing*, then updates the smoothed values *y* in the half of the window closest to the edge.

cnvlib.smoothing.**outlier_iqr**(*a*, *c=1.5*)

Detect outliers as a multiple of the IQR from the median.

By convention, "outliers" are points more than 1.5 * IQR from the median, and "extremes" or extreme outliers are those more than 3.0 * IQR.

cnvlib.smoothing.**outlier_mad_median**(*a*)

MAD-Median rule for detecting outliers.

Returns: a boolean array of the same size, where outlier indices are True.

X_i is an outlier if:

```
 | X_i - M |
_____________  > K ~= 2.24

 MAD / 0.6745
```

where $K = sqrt(X^2_{0.975,1})$, the square root of the 0.975 quantile of a chi-squared distribution with 1 degree of freedom.

This is a very robust rule with the highest possible breakdown point of 0.5.

See:

- Davies & Gather (1993) The Identification of Multiple Outliers.
- Rand R. Wilcox (2012) Introduction to robust estimation and hypothesis testing. Ch.3: Estimating measures of location and scale.

cnvlib.smoothing.**rolling_median**(*x*, *width*)

Rolling median.

Contributed by Peter Otten to comp.lang.python.

Source: https://bitbucket.org/janto/snippets/src/tip/running_median.py https://groups.google.com/d/msg/comp.lang.python/0OAR glW4t6gJ

cnvlib.smoothing.**smooth_genome_coverages**(*probes*, *smooth_func*, *width*)

Fit a trendline through probe coverages, handling chromosome boundaries.

Returns an array of smoothed coverage values, calculated with *smooth_func* and *width*, equal in length to *probes*.

cnvlib.smoothing.**smoothed**(*x*, *width*, *do_fit_edges=False*)

Smooth the values in *x* with the Kaiser windowed filter.

See: http://en.wikipedia.org/wiki/Kaiser_window

Parameters:

x [array-like] 1-dimensional numeric data set.

width [float] Fraction of x's total length to include in the rolling window (i.e. the proportional window width), or the integer size of the window.

CHAPTER 5

Citation

We are in the process of publishing a manuscript describing CNVkit. If you use this software in a publication, for now, please cite our preprint manuscript by DOI, like so:

> Eric Talevich, A. Hunter Shain, Thomas Botton, Boris C. Bastian (2014) CNVkit: Copy number detection and visualization for targeted sequencing using off-target reads. *bioRxiv* doi: http://dx.doi.org/10.1101/010876

A poster presentation can be viewed at F1000 Research.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index