CNVkit Documentation

Release 0.3.5

Eric Talevich

August 19, 2015

Contents

1 Quick start **3**
- 1.1 Install CNVkit . . . 3
- 1.2 Download the reference genome . . . 3
- 1.3 Map sequencing reads to the reference genome . . . 4
- 1.4 Build a reference from normal samples and infer tumor copy ratios . . . 4
- 1.5 Process more tumor samples . . . 5

2 Command line usage **7**
- 2.1 Copy number pipeline . . . 8
- 2.2 Plots and graphics . . . 11
- 2.3 Text and tabular reports . . . 12
- 2.4 Compatibility and other I/O . . . 14
- 2.5 Additional scripts . . . 14

3 Python API **17**
- 3.1 Python API (cnvlib package) . . . 17

4 Citation **33**

5 Indices and tables **35**

Python Module Index **37**

Author Eric Talevich

Contact eric.talevich@ucsf.edu

Source code http://github.com/etal/cnvkit

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.

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-10kb.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, 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-10kb.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-10kb.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-10kb.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/
```

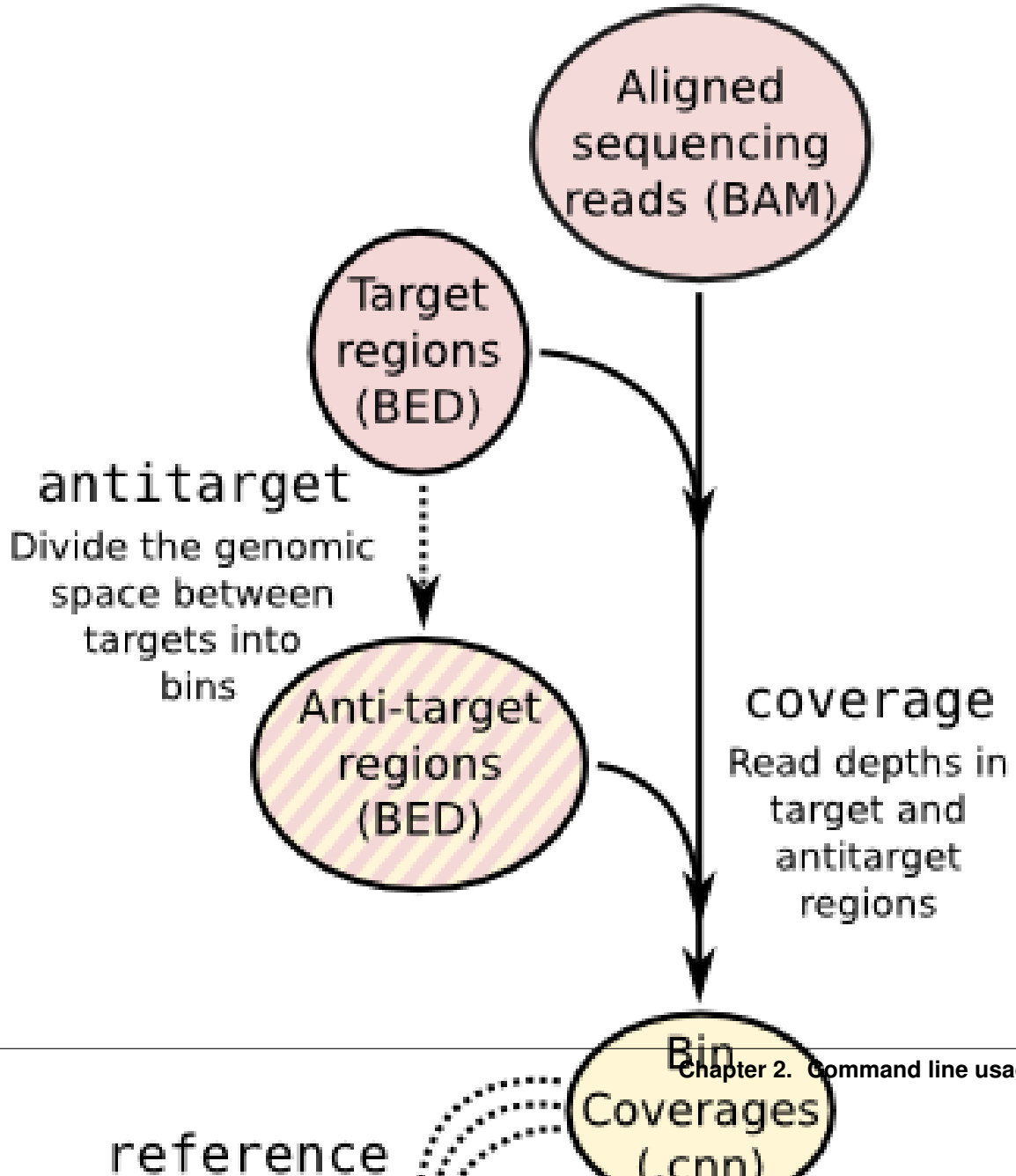
The coordinates of the target and antitarget bins, the gene names for the targets, and the GC and RepeatMasker information for bias corrections are automatically extracted from the reference .cnn file you've built.

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 pipeline



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 antitarget

Derive a background/"antitarget" BED file from a "target" BED file that lists the chromosomal coordinates of the tiled regions used for targeted resequencing.

```
cnvkit.py antitarget Tiled.bed -g data/access-10000.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, and can be computed for other genomes with the included script `genome2access.py`.

To use CNVkit on **amplicon** sequencing data instead of **hybrid capture** – 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 antitarget Tiled.bed -g Tiled.bed -o Background.bed
cnvkit.py batch ... -t Tiled.bed -g Tiled.bed ...
```

This results in empty ".antitarget.cnn" files which CNVkit will handle safely from version 0.3.4 onward. 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.

2.1.3 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
```

About those BAM files:

- **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.
- **If you've prebuilt the 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.

2.1.4 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.

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.

About the FASTA index file:

- 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.

2.1.5 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
```

2.1.6 segment

Infer discrete copy number segments from the given coverage table. By default this uses the circular binary segmentation algorithm (CBS), but with the '-m haar' option, the faster but less accurate HaarSeg algorithm can be used instead.

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

The output table of copy number segments (.cns) is essentially the same tabular format as the other .cnn and .cnr files.

2.2 Plots and graphics

2.2.1 scatter

Plot probe log2 coverages and segmentation calls together.

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

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

```
cnvkit.py scatter Sample.cnr -s Sample.cns -r chr7
cnvkit.py scatter Sample.cnr -s Sample.cns -r BRAF
cnvkit.py scatter Sample.cnr -s Sample.cns -r chr7:140434347-140624540
```

In the latter two cases, the `--width` (`-w`) argument determines the size of the chromosomal regions to show flanking the selected region.

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, below.)

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

The probe copy number values can also be plotted without segmentation calls:

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

This can be useful if CBS is unavailable, or for viewing the raw, un-corrected coverages 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 CBS 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 raw probes (.cnn, .cnr) or segments (.cns)) on chromosomes as a diagram. If both the raw probes and segmentation calls are given, show them side-by-side on each chromosome (segments on the left side, probes on the right side).

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

2.2.4 heatmap

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

The segmentation calls alone will render much faster, and will probably be more useful to look at.

```
cnvkit.py heatmap *.cns
cnvkit.py heatmap *.cnr  # Slow!
```

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.

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 -y -m 5
```

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 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 .6 (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, which is amenable to further processing by scripts and standard Unix tools such as `grep`, `sort`, `cut` and `awk`.

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
```

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.

2.4 Compatibility and other I/O

2.4.1 import-picard

Convert Picard CalculateHsMetrics 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.

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
```

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
```

Also note that the individual `.cnr` and `.cnn` files can be loaded directly by the commercial program Biodiscovery Nexus Copy Number, specifying the "basic" input format.

2.5 Additional scripts

refFlat2bed.py Generate a BED file of the genes or exons in the reference genome given in UCSC refFlat.txt format. 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, better results are expected from CNVkit if the true targeted intervals can be provided.

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

CHAPTER 3

Python API

3.1 Python API (cnvlib package)

3.1.1 Module `cnvlib` contents

`cnvlib.`**`read`**(*fname*)
Parse a file as a copy number or copy ratio table (.cnn, .cnr).

3.1.2 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*, *chromosomes*, *starts*, *ends*, *genes*, *coverages*, *gc=None*, *rmask=None*, *spread=None*, *weight=None*, *probes=None*)
Bases: `object`

An array of genomic intervals, treated like aCGH probes.

`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(*mode=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

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

Any optional columns must match between both arrays.

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()

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

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_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=0.6*, *min_probes=1*)
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*)
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*)

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.**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_normal=False*, *chr_x=None*, *verbose=True*)
Guess whether a sample is female from chrX relative coverages.

Recommended cutoff values: -0.4 – raw target data, not yet corrected +0.7 – 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_normal=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**(*probe_pset*, *seg_pset*, *threshold*, *outfname*, *male_normal*)
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.**cna_absolutes**(*cnarr*, *ploidy*, *purity*, *is_reference_male*, *is_sample_female*)
Calculate absolute copy number values from segment or bin log2 ratios.

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

cnvlib.export.**export_freebayes**(*sample_fname*, *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.**rescale_copy_ratios**(*cnarr*, *purity=None*, *ploidy=2*, *is_sample_female=None*, *is_reference_male=True*)
Rescale segment copy ratio values given a known tumor purity.

cnvlib.export.**round_to_integer**(*ncopies*, *half_is_zero=True*, *rounding_error=1e-07*)
Round an absolute estimate of copy number to a positive integer.

half_is_zero indicates the hack of encoding 0 copies (complete loss) as a half-copy in log2 scale (e.g. log2-ratio value of -2.0 for diploid) to avoid domain errors when log-transforming. If *half_is_zero*, a half-copy will be rounded down to zero rather than up to 1 copy.

cnvlib.export.**row_to_probe_coverage**(*row*)
Repack a parsed row into a ProbeInfo instance and coverage value.

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_arr*, *min_weight=1e-05*)
Calculate weights for each bin.

Weights are derived from the "spread" column of the reference. In future, deviations within a rolling in the sample array may also be considered.

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 'l', 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*)
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

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.**echo**(**words*)

cnvlib.ngfrills.**ensure_bam_index**(*bam_fname*)
Ensure a BAM file is indexed, to enable fast traversal & lookup.

cnvlib.ngfrills.**ensure_bam_sorted**(*bam_fname*, *by_name=False*, *span=50*)
Test if the reads in a BAM file are sorted as expected.

by_name=True: reads are expected to be sorted by query name. Consecutive read IDs are in alphabetical order, and read pairs appear together.

by_name=False: reads are sorted by position. Consecutive reads have increasing position.

cnvlib.ngfrills.**ensure_fasta_index**(*fasta_fname*)
Ensure a FASTA file is indexed for samtools, to enable fast lookup.

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.**fasta_extract_regions**(*fa_fname*, *intervals*)
Extract an iterable of regions from an indexed FASTA file.

Input: indexed FASTA file name; iterable of (seq_id, start, end) (1-based) Output: iterable of string sequences.

cnvlib.ngfrills.**filter_vcf_lines**(*vcf_fname*, *min_depth*, *skip_hom*)

cnvlib.ngfrills.**group_bed_tracks**(*bedfile*)
Group the parsed rows in a BED file by track.

Yields (track_name, iterable_of_lines), much like itertools.groupby.

cnvlib.ngfrills.**is_newer_than**(*target_fname*, *orig_fname*)

cnvlib.ngfrills.**load_vcf**(*fname*, *min_depth=1*, *skip_hom=True*)
Parse SNV coordinates from a VCF file; group by chromosome.

Returns a dict of: {chrom: position, zygosity, alt allele frequency}

cnvlib.ngfrills.**parse_bed**(*fname*, *coord_only*, *keep_strand*)
Parse a BED file.

A BED file has these columns: chromosome, start position, end position, [name, strand, other stuff...]

Counting is from 0.

Sets of regions are separated by "track" lines. This function stops iteration after encountering a track line other than the first one in the file.

cnvlib.ngfrills.**parse_bed_track**(*line*)
Parse the "name" field of a BED track definition line.

Example: track name=146793_BastianLabv2_P2_target_region description="146793_BastianLabv2_P2_target_region"

cnvlib.ngfrills.**parse_interval_list**(*fname*, *coord_only*, *keep_strand*)
Parse a Picard-compatible interval list.

Expected tabular columns: chromosome, start position, end position, strand, region name

Counting is from 1.

cnvlib.ngfrills.**parse_regions**(*fname*, *coord_only=False*, *keep_strand=False*)
Parse regions in any of the expected file formats.

Iterates over tuples of the tabular contents. Header lines are skipped.

Start and end coordinates are base-0, half-open.

If coord_only, yield triplets of (chrom, start, end). Otherwise, yield quads of (chrom, start, end, name).

cnvlib.ngfrills.**parse_text_coords**(*fname*, *coord_only*, *keep_strand*)
Parse text coordinates: chrom:start-end

Text coordinates are assumed to be counting from 1.

cnvlib.ngfrills.**read_fasta_index**(*fasta_fname*)
Load a FASTA file's index.

Returns a dict of: {seq_id: (length, offset, chars_per_line, bytes_per_line), ...}

The index file contains, in one row per sequence, tab-separated columns:

- sequence identifier
- length
- offset of the first sequence character in the file
- number of characters per line
- number of bytes per line (including the end-of-line character)

With this information, we can easily compute the byte offset of the i-th character of a sequence in a file by looking at its index record. We skip to this byte offset in the file and from there, we can read the necessary sequence characters.

See: http://trac.seqan.de/wiki/Tutorial/IndexedFastaIO:

cnvlib.ngfrills.**report_bad_line**(*line_parser*)

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.**sniff_num_columns**(*bed_fname*)
Detect the number of columns in a BED/interval file.

Guidance: 3 cols => coordinates only; 5 cols => intervals file (coordinates, strand, name); otherwise => Full or extended BED format

cnvlib.ngfrills.**sniff_region_format**(*fname*)
Guess whether the file format is BED, Picard interval list, or text.

Returns a tuple of the format name (str) or None if the file is empty.

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

cnvlib.plots.**plot_loh**(*axis*, *chrom_snvs*, *chrom_sizes*, *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*, *has_genome*, *is_male_normal*)
Calculate the median coverage of each probe across multiple samples.

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

Returns: A single CopyNumArray summarizing the coverages of the input samples, including each probe'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.**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*, *rlib-path=None*)
Infer copy number segments from the given coverage table.

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/0OARglW4t6gJ

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 4

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 recent poster presentation is also available on F1000 Posters.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

C

cnvlib, 17
cnvlib.antitarget, 20
cnvlib.cnarray, 17
cnvlib.commands, 19
cnvlib.core, 21
cnvlib.coverage, 22
cnvlib.diagram, 22
cnvlib.export, 23
cnvlib.fix, 25
cnvlib.importers, 25
cnvlib.metrics, 26
cnvlib.ngfrills, 27
cnvlib.params, 29
cnvlib.plots, 29
cnvlib.reference, 30
cnvlib.reports, 31
cnvlib.segmentation, 31
cnvlib.smoothing, 31

A

apply_async() (cnvlib.commands.SerialPool method), 19
apply_weights() (in module cnvlib.fix), 25
assert_equal() (in module cnvlib.core), 21

B

bam_total_reads() (in module cnvlib.coverage), 22
batch_make_reference() (in module cnvlib.commands), 19
batch_run_sample() (in module cnvlib.commands), 19
batch_write_coverage() (in module cnvlib.commands), 19
bc_chromosome_draw_label() (in module cnvlib.diagram), 22
bc_organism_draw() (in module cnvlib.diagram), 22
bed2probes() (in module cnvlib.reference), 30
bedcov() (in module cnvlib.coverage), 22
biweight_location() (in module cnvlib.metrics), 26
biweight_midvariance() (in module cnvlib.metrics), 26
build_chrom_diagram() (in module cnvlib.diagram), 23
by_bin() (cnvlib.cnarray.CopyNumArray method), 17
by_chromosome() (cnvlib.cnarray.CopyNumArray method), 17
by_gene() (cnvlib.cnarray.CopyNumArray method), 17
by_segment() (cnvlib.cnarray.CopyNumArray method), 17

C

calculate_gc_lo() (in module cnvlib.reference), 30
calculate_theta_fields() (in module cnvlib.export), 23
call_quiet() (in module cnvlib.ngfrills), 27
center_all() (cnvlib.cnarray.CopyNumArray method), 18
center_by_window() (in module cnvlib.fix), 25
check_inputs() (in module cnvlib.smoothing), 31
check_unique() (in module cnvlib.core), 21
chrom (cnvlib.export.ProbeInfo attribute), 23
chromosome (cnvlib.cnarray.CopyNumArray attribute), 18
chromosome_sizes() (in module cnvlib.plots), 29
close() (cnvlib.commands.SerialPool method), 19
cna_absolutes() (in module cnvlib.export), 23
cnvlib (module), 17
cnvlib.antitarget (module), 20
cnvlib.cnarray (module), 17
cnvlib.commands (module), 19
cnvlib.core (module), 21
cnvlib.coverage (module), 22
cnvlib.diagram (module), 22
cnvlib.export (module), 23
cnvlib.fix (module), 25
cnvlib.importers (module), 25
cnvlib.metrics (module), 26
cnvlib.ngfrills (module), 27
cnvlib.params (module), 29
cnvlib.plots (module), 29
cnvlib.reference (module), 30
cnvlib.reports (module), 31
cnvlib.segmentation (module), 31
cnvlib.smoothing (module), 31
combine_probes() (in module cnvlib.reference), 30
copy() (cnvlib.cnarray.CopyNumArray method), 18
CopyNumArray (class in cnvlib.cnarray), 17
coverage (cnvlib.cnarray.CopyNumArray attribute), 18
create_chrom_ids() (in module cnvlib.export), 23
create_diagram() (in module cnvlib.diagram), 23
create_heatmap() (in module cnvlib.commands), 19
create_loh() (in module cnvlib.commands), 19
cvg2rgb() (in module cnvlib.plots), 29

D

do_antitarget() (in module cnvlib.commands), 20
do_breaks() (in module cnvlib.commands), 20
do_coverage() (in module cnvlib.commands), 20
do_fix() (in module cnvlib.commands), 20
do_gainloss() (in module cnvlib.commands), 20
do_reference() (in module cnvlib.commands), 20
do_reference_flat() (in module cnvlib.commands), 20
do_scatter() (in module cnvlib.commands), 20
do_segmentation() (in module cnvlib.segmentation), 31
do_targets() (in module cnvlib.commands), 20
drop_extra_columns() (cnvlib.cnarray.CopyNumArray method), 18

E

echo() (in module cnvlib.ngfrills), 27
edge_gain() (in module cnvlib.fix), 25
edge_loss() (in module cnvlib.fix), 25
end (cnvlib.cnarray.CopyNumArray attribute), 18
end (cnvlib.export.ProbeInfo attribute), 23
ensure_bam_index() (in module cnvlib.ngfrills), 27
ensure_bam_sorted() (in module cnvlib.ngfrills), 27
ensure_fasta_index() (in module cnvlib.ngfrills), 27
ensure_path() (in module cnvlib.ngfrills), 27
ests_of_scale() (in module cnvlib.metrics), 26
export_freebayes() (in module cnvlib.export), 23
export_nexus_basic() (in module cnvlib.export), 24
export_seg() (in module cnvlib.export), 24
export_theta() (in module cnvlib.export), 24
extend() (cnvlib.cnarray.CopyNumArray method), 18

F

fasta_extract_regions() (in module cnvlib.ngfrills), 27
fbase() (in module cnvlib.core), 21
filter_column() (in module cnvlib.coverage), 22
filter_read() (in module cnvlib.coverage), 22
filter_vcf_lines() (in module cnvlib.ngfrills), 27
find_background_regions() (in module cnvlib.antitarget), 20
find_picard_files() (in module cnvlib.importers), 25
fit_edges() (in module cnvlib.smoothing), 31
fmt_cdt() (in module cnvlib.export), 24
fmt_gct() (in module cnvlib.export), 24
fmt_jtv() (in module cnvlib.export), 24
fmt_multi() (in module cnvlib.export), 24
fmt_vcf() (in module cnvlib.export), 24
from_rows() (cnvlib.cnarray.CopyNumArray class method), 18

G

gene (cnvlib.cnarray.CopyNumArray attribute), 18
gene (cnvlib.export.ProbeInfo attribute), 23
gene_coords_by_name() (in module cnvlib.plots), 29
gene_coords_by_range() (in module cnvlib.plots), 29
get_background() (in module cnvlib.antitarget), 20
get_breakpoints() (in module cnvlib.reports), 31
get_fasta_stats() (in module cnvlib.reference), 30
get_gene_intervals() (in module cnvlib.reports), 31
get_relative_chrx_cvg() (in module cnvlib.core), 21
group_bed_tracks() (in module cnvlib.ngfrills), 27
group_by_genes() (in module cnvlib.reports), 31
group_coords() (in module cnvlib.antitarget), 21
guess_chr_x() (in module cnvlib.core), 21
guess_chromosome_regions() (in module cnvlib.antitarget), 21
guess_xx() (in module cnvlib.core), 21

I

import_seg() (in module cnvlib.importers), 26
in_range() (cnvlib.cnarray.CopyNumArray method), 18
interquartile_range() (in module cnvlib.metrics), 26
interval_coverages() (in module cnvlib.coverage), 22
interval_coverages_count() (in module cnvlib.coverage), 22
interval_coverages_pileup() (in module cnvlib.coverage), 22
is_newer_than() (in module cnvlib.ngfrills), 28

J

join() (cnvlib.commands.SerialPool method), 19

L

label (cnvlib.export.ProbeInfo attribute), 23
labels() (cnvlib.cnarray.CopyNumArray method), 18
limit() (in module cnvlib.plots), 29
load_adjust_coverages() (in module cnvlib.fix), 25
load_targetcoverage_csv() (in module cnvlib.importers), 26
load_vcf() (in module cnvlib.ngfrills), 28

M

make_edge_sorter() (in module cnvlib.fix), 25
mask_bad_probes() (in module cnvlib.reference), 30
match_ref_to_probes() (in module cnvlib.fix), 25
median_absolute_deviation() (in module cnvlib.metrics), 27
merge_rows() (in module cnvlib.export), 24
merge_samples() (in module cnvlib.export), 24

O

outlier_iqr() (in module cnvlib.smoothing), 31
outlier_mad_median() (in module cnvlib.smoothing), 31

P

parse_args() (in module cnvlib.commands), 20
parse_bed() (in module cnvlib.ngfrills), 28
parse_bed_track() (in module cnvlib.ngfrills), 28
parse_interval_list() (in module cnvlib.ngfrills), 28
parse_range() (in module cnvlib.plots), 29
parse_regions() (in module cnvlib.ngfrills), 28
parse_text_coords() (in module cnvlib.ngfrills), 28
parse_theta_results() (in module cnvlib.importers), 26
parse_tsv() (in module cnvlib.core), 21
partition_by_chrom() (in module cnvlib.plots), 29
pick_pool() (in module cnvlib.commands), 20
plot_chromosome() (in module cnvlib.plots), 30
plot_genome() (in module cnvlib.plots), 30
plot_loh() (in module cnvlib.plots), 30
plot_x_dividers() (in module cnvlib.plots), 30
probe_center() (in module cnvlib.plots), 30

probe_deviations_from_segments() (in module cnvlib.metrics), 27
ProbeInfo (class in cnvlib.export), 23

Q

q_n() (in module cnvlib.metrics), 27

R

rbase() (in module cnvlib.core), 21
read() (cnvlib.cnarray.CopyNumArray class method), 18
read() (in module cnvlib), 17
read_fasta_index() (in module cnvlib.ngfrills), 28
reference2regions() (in module cnvlib.reference), 30
region_depth_count() (in module cnvlib.coverage), 22
report_bad_line() (in module cnvlib.ngfrills), 29
rescale_copy_ratios() (in module cnvlib.export), 24
rolling_median() (in module cnvlib.smoothing), 32
round_to_integer() (in module cnvlib.export), 24
row2label() (in module cnvlib.cnarray), 19
row_to_probe_coverage() (in module cnvlib.export), 24

S

safe_write() (in module cnvlib.ngfrills), 29
segments2freebayes() (in module cnvlib.export), 24
select() (cnvlib.cnarray.CopyNumArray method), 18
SerialPool (class in cnvlib.commands), 19
shift_xx() (in module cnvlib.core), 21
shuffle() (cnvlib.cnarray.CopyNumArray method), 18
smooth_genome_coverages() (in module cnvlib.smoothing), 32
smoothed() (in module cnvlib.smoothing), 32
sniff_num_columns() (in module cnvlib.ngfrills), 29
sniff_region_format() (in module cnvlib.ngfrills), 29
sort() (cnvlib.cnarray.CopyNumArray method), 18
sorter_chrom() (in module cnvlib.core), 21
sorter_chrom_at() (in module cnvlib.core), 22
squash_genes() (cnvlib.cnarray.CopyNumArray method), 19
squash_segments() (in module cnvlib.segmentation), 31
start (cnvlib.cnarray.CopyNumArray attribute), 19
start (cnvlib.export.ProbeInfo attribute), 23

T

temp_write_text() (in module cnvlib.ngfrills), 29
test_loh() (in module cnvlib.plots), 30
to_rows() (cnvlib.cnarray.CopyNumArray method), 19

U

unpipe_name() (in module cnvlib.importers), 26

W

warn_bad_probes() (in module cnvlib.reference), 30
write() (cnvlib.cnarray.CopyNumArray method), 19
write_tsv() (in module cnvlib.core), 22