Copy number calling and SNV classification using targeted short read sequencing

Markus Riester[1]

[1]Novartis Institutes for BioMedical Research, Cambridge, MA

October 28, 2016

Abstract

PureCN is a purity and ploidy aware copy number caller for cancer samples inspired by the *ABSOLUTE* algorithm [**?**]. It was designed for hybrid capture sequencing data, especially with medium-sized targeted gene panels without matching normal samples in mind (matched whole exome data is of course supported).

It can be used to supplement existing normalization and segmentation algorithms, i.e. the software can start from BAM files, from target-level coverage data, from copy number log-ratios or from already segmented data. If the correct purity and ploidy solution was identified, *PureCN* can also help in classifying variants as germline vs. somatic or clonal vs. sub-clonal.

PureCN was further designed to integrate well with industry standard pipelines [**?**], but it is straightforward to generate input data from other pipelines.

Package

PureCN 1.2.3

Contents

1 Quick start

This tutorial will demonstrate on a toy example how we recommend running *PureCN* on targeted sequencing data. To estimate tumor purity, we jointly utilize both target-level[1] coverage data and allelic fractions of single nucleotide variants (SNVs), inside - and optionally outside - the targeted regions. Knowledge of purity will in turn allow us to accurately (i) infer integer copy number and (ii) classify variants (somatic vs. germline, mono-clonal vs. sub-clonal, heterozygous vs. homozygous etc.).

[1] The captured genomic regions, e.g. exons.

This requires 3 basic input files:

1. A VCF file containing germline SNPs and somatic mutations. Somatic status is not required in case the variant caller was run without matching normal sample.
2. The tumor BAM file.
3. A BAM file from a normal control sample, either matched or process-matched.

In addition, we need to know a little bit more about the assay. This is the annoying step, since here you need to provide some information. Most importantly, we need to know the positions of all targets. Then we need to correct for GC-bias, for which we need GC-content for each target. Optionally, if gene-level calls are wanted, we also need for each target a gene symbol. To obtain best results, we can finally use a pool of normal samples to automatically learn more about our assay and its biases and common artifacts.

The next sections will show how to do all this with *PureCN* alone or with the help of *GATK* and/or existing copy number pipelines. We tried to make this as pain-free as possible.

2 Basic input files

2.1 Single nucleotide variants

Germline SNPs and somatic mutations are expected in a single VCF file. At the bare minimum, this VCF should contain read depths of reference and alt alleles in an AD genotype field and a DB info flag for dbSNP membership. If a matched normal is available, then somatic status information is currently expected in a SOMATIC info flag in the VCF. The *VariantAnnotation* package provides examples how to add info fields to a VCF in case the used variant caller does not add this flag.

VCF files generated by *MuTect* [**?**] should work well and in general require no post-processing. *PureCN* can handle *MuTect* VCF files generated in both single and matched normal mode.

2.2 Coverage data

For the default segmentation function provided by *PureCN*, the algorithm first needs to calculate log-ratios of tumor vs. normal control coverage. Coverage data needs to be provided in *GATK DepthOfCoverage* format[2]:

[2] *GATK* will generate additional columns and *PureCN* will ignore these when provided

```
Target  total_coverage  average_coverage
chr1:69091-70009    0   0   0
chr1:367659-368598  6358    9.25
chr1:621096-622035  6294    9.16
```

The intervals define the captured genomic regions (targets) and are provided by the manufacturer of your capture kit[3]. Default parameters assume that these intervals do NOT include a "padding" to include flanking regions of targets. *PureCN* can include variants in flanking regions if the variant caller was run with interval padding (See Sections **??** and **??**).

[3] While *PureCN* can use a pool of normal samples to learn which intervals are reliable and which not, it is highly recommended to provide the correct intervals. Garbage in, garbage out.

It is recommended to GC-normalize coverage (how this done is described later in Section **??**). The GC-content for each target is expected in *GATK GCContentByInterval* format:

```
Target    gc_bias    Gene
chr1:69091-70009    0.427638737758433    OR4F5
chr1:367659-368598    0.459574468085106    OR4F29
chr1:621096-622035    0.459574468085106    OR4F3
```

The `Gene` column is optional and only required for providing gene-level copy number and LOH calls. This information should be available in the technical documentation of your capture kit. Simplistic example code for annotating target intervals with gene symbols is given in the Appendix of this vignette for testing purposes.

2.3 Generating coverage data without *GATK*

The `calculateBamCoverageByInterval` function can be used to generate the required coverage data from BAM files:

```
bam.file <- system.file("extdata", "ex1.bam", package="PureCN",
    mustWork = TRUE)
interval.file <- system.file("extdata", "ex1_intervals.txt",
    package="PureCN", mustWork = TRUE)

calculateBamCoverageByInterval(bam.file=bam.file,
 interval.file=interval.file, output.file="ex1_coverage.txt")
```

To calculate GC-content, *PureCN* provides the `calculateGCContentByInterval` function:

```
interval.file <- system.file("extdata", "ex2_intervals.txt",
        package = "PureCN", mustWork = TRUE)
```

```
reference.file <- system.file("extdata", "ex2_reference.fa",
    package = "PureCN", mustWork = TRUE)
calculateGCContentByInterval(interval.file, reference.file,
    output.file = "ex2_gc_file.txt")
```

2.4 Third-party segmentation tools

PureCN integrates well with existing copy number pipelines. Instead of coverage data, the user then needs to provide either already segmented data or a wrapper function. This is described in Section **??**.

2.5 Example data

We now load a few example files that we will use throughout this tutorial:

```
normal.coverage.file <- system.file("extdata", "example_normal.txt",
    package="PureCN")
normal2.coverage.file <- system.file("extdata", "example_normal2.txt",
    package="PureCN")
normal.coverage.files <- c(normal.coverage.file, normal2.coverage.file)
tumor.coverage.file <- system.file("extdata", "example_tumor.txt",
    package="PureCN")
seg.file <- system.file("extdata", "example_seg.txt",
    package = "PureCN")
vcf.file <- system.file("extdata", "example_vcf.vcf", package="PureCN")
gc.gene.file <- system.file("extdata", "example_gc.gene.file.txt",
    package="PureCN")
```

3 GC-bias

The algorithm works best when the coverage files are GC-normalized. We can easily create GC-normalized coverage files:

```
correctCoverageBias(normal.coverage.file, gc.gene.file,
    "example_normal_loess.txt")
```

All the following steps in this vignette assume that the coverage data are GC-normalized. The example coverage files are already GC-normalized. Section **??** presents a convenient command line script for generating GC-normalized coverage data from BAM files or from *GATK* coverage files.

4 Pool of normals

4.1 Selection of normals for log-ratio calculation

For calculating copy number log-ratios of tumor vs. normal, *PureCN* requires coverage from a process-matched normal sample. Using a normal that was sequenced using a similar, but not identical assay, rarely works, since differently covered genomic regions result in too many log-ratio outliers. This section describes how to identify a good process-matched normal in case no matched normal is available or in case the matched normal has low or uneven coverage.

The `createNormalDatabase` function builds a database of coverage files:

```
normalDB <- createNormalDatabase(normal.coverage.files)

## Allosome coverage appears to be missing, cannot determine sex.
## Allosome coverage appears to be missing, cannot determine sex.

# serialize, so that we need to do this only once for each assay
saveRDS(normalDB, file="normalDB.rds")
```

Again, please make sure that all coverage files were GC-normalized prior to building the database (Section **??**). Internally, `createNormalDatabase` determines the sex of the samples and trains a PCA that is later used for clustering a tumor file with all normal samples in the database. This clustering is performed by the `findBestNormal` function:

```
normalDB <- readRDS("normalDB.rds")
# get the best normal
best.normal.coverage.file <- findBestNormal(tumor.coverage.file,
    normalDB)
```

This function can also return multiple normal files that can be averaged into a single pool:

```
# get the best 2 normals and average them
best.normal.coverage.files <- findBestNormal(tumor.coverage.file,
    normalDB, num.normals=2)
pool <- poolCoverage(lapply(best.normal.coverage.files,
    readCoverageGatk), remove.chrs=c('chrX', 'chrY'))
```

Pooling is only recommended when the coverage in normals is significantly lower than in tumor. Otherwise the PCA will typically do a good job in selecting a normal with decent coverage and similar biases compared to tumor. But it is worth experimenting with different strategies using the `plotBestNormal` function. Note that this example removes coverage from sex chromosomes; if the normal database contains a sufficient number of samples with matching sex, `findBestNormal` will return only normal samples with matching sex.

4.2 Artifact filtering

It is important to remove as many artifacts as possible, since low ploidy solutions are typically punished more by artifacts than high ploidy solutions. High ploidy solutions are complex and usually find ways of explaining artifacts reasonably well. This following steps in this section are optional, but recommended especially when matched normals are not available.

We first use coverage data of normal samples to estimate the expected variance in coverage per target:

```
target.weight.file <- "target_weights.txt"
createTargetWeights(tumor.coverage.file, normal.coverage.files,
    target.weight.file)

## Loading coverage data...

## Average coverage:  100X (tumor), 99X (normal).

## Average coverage:  100X (tumor), 43X (normal).
```

This function calculates target-level copy number log-ratios for all normal samples provided in the `normal.coverage.files` argument. Assuming that all normal samples are in general diploid, an unusual high variance in log-ratio is indicative of an target with either common germline alterations or frequent artifacts; high or low copy number log-ratios in these targets are unlikely measuring somatic copy number events. For the log-ratio calculation, we provide a coverage file that is used as tumor in the log-ratio calculation. The corresponding `tumor.coverage.file` argument can also be an array of coverage files, in which case the target coverage variance is averaged over all provided tumor files. As long as the coverage of the tumor files is even, only 1 or 2 tumor files are necessary, however.

We can also use the pool of normals to find SNPs with biased allelic fractions in low quality regions (very significantly different from 0.5 for heterozygous SNPs). If a matching normal is provided, most variants in low quality regions with biased allelic fractions should be automatically ignored. Note that this is not meant to model non-reference bias leading in expected allelic ratio only slightly below 0.5 (typically around 0.48 on average). This bias is typically not strong enough to influence selection of SNV states.

```
snp.blacklist <- c("SNP_blacklist.csv", "SNP_blacklist_segmented.csv")
recreateBlacklists <- FALSE

if (recreateBlacklists) {
    mutect.normal.files <- dir("poolofnormals", pattern="vcf$",
        full.names=TRUE)
    snp.bl <- createSNPBlacklist(mutect.normal.files)
    write.csv(snp.bl[[1]], file=snp.blacklist[1])
    write.csv(snp.bl[[2]], file=snp.blacklist[2], row.names=FALSE,
       quote=FALSE)
}
```

Finally, we recommend running *MuTect* with a pool of normal samples to filter common sequencing errors and alignment artifacts from the VCF. *MuTect* requires a single VCF containing all normal samples, for example generated by the *GATK CombineVariants* tool. It is possible to provide *PureCN* this combined VCF as well; it might help the software correcting non-reference read mapping biases. This is described in the `setMappingBiasVcf` documentation.

4.3 Artifact filtering without a pool of normals

By default, *PureCN* will exclude targets with coverage below 15X from segmentation. For SNVs, the same 15X cutoff is applied. *MuTect* applies more sophisticated artifact tests and flags suspicious variants. If *MuTect* was run in matched normal mode, then both potential artifacts and germline variants are rejected, that means we cannot just filter by the PASS/REJECT *MuTect* flags. The `filterVcfMuTect` function optionally reads the *MuTect 1.1.7* stats file and will keep germline variants, while removing potential artifacts. Without the stats file, *PureCN* will use only the filters based on read depths as defined in `filterVcfBasic`. Both functions are automatically called by *PureCN*, but can be easily modified and replaced if necessary.

Instead of using a pool of normals to find SNPs with biased allelic fractions, we can also use a BED file to blacklist regions. For example the segmental duplications and simple repeats tracks from the UCSC. This is again recommended when matching normals are not available.

```
# Instead of using a pool of normals to find low quality regions,
# we use suitable BED files, for example from the UCSC genome browser.

# We do not download these in this vignette to avoid build failures
# due to internet connectivity problems.
downloadFromUCSC <- FALSE
if (downloadFromUCSC) {
    library(rtracklayer)
    mySession <- browserSession("UCSC")
    genome(mySession) <- "hg19"
    tbl.segmentalDups <- getTable( ucscTableQuery(mySession,
        track="Segmental Dups", table="genomicSuperDups"))
    tbl.simpleRepeats <- getTable( ucscTableQuery(mySession,
        track="Simple Repeats", table="simpleRepeat"))
    write.table(tbl.segmentalDups[,-1],
        file="hg19_segmentalDuplications.txt", sep="\t",
        row.names=FALSE)
    write.table(tbl.simpleRepeats[,-1],
        file="hg19_simpleRepeats.txt", sep="\t",
        row.names=FALSE)
}

snp.blacklist <- c('hg19_segmentalDuplications.txt',
```

```
        'hg19_simpleRepeats.txt')
```

5 Recommended run

Finally, we can run *PureCN* with all that information:

```
ret <-runAbsoluteCN(normal.coverage.file=normal.coverage.file,
    tumor.coverage.file=tumor.coverage.file, vcf.file=vcf.file,
    genome="hg19", sampleid='Sample1',
    gc.gene.file=gc.gene.file,
# args.filterVcf=list(snp.blacklist=snp.blacklist,
# stats.file=mutect.stats.file),
    args.filterTargets=list(normalDB=normalDB),
    args.segmentation=list(target.weight.file=target.weight.file),
    post.optimize=FALSE, plot.cnv=FALSE, verbose=FALSE)
```

The `normal.coverage.file` argument points to a coverage file obtained from either a matched or a process-matched normal sample, but can be also a small pool of best normals (Section **??**). The files specified in `args.filterVcf` help *PureCN* filtering SNVs more efficiently for artifacts as described in Sections **??** and **??**. The `normalDB` argument (Section **??**) in `args.filterTargets` allows the segmentation function to skip targets with low coverage in the pool of normals. If possible, these files should be generated and provided. The `post.optimize` flag will increase the runtime by about a factor of 4-5, but might return slightly more accurate purity estimates. For high quality whole exome data, this is typically not necessary for copy number calling (but might be for variant classification, see Section **??**). The `plot.cnv` argument allows the segmentation function to generate additional plots if set to `TRUE`. Finally, `verbose` outputs important and helpful information about all the steps performed and is therefore set to `TRUE` by default.

We now create a few output files:

```
file.rds <- 'Sample1_PureCN.rds'
saveRDS(ret, file=file.rds)
pdf('Sample1_PureCN.pdf', width=10, height=12)
plotAbs(ret, type='all')
dev.off()
```

```
## pdf
##   2
```

The RDS file now contains the serialized return object of the `runAbsoluteCN` call. The PDF contains helpful plots for all local minima, sorted by likelihood. The first plot in the generated PDF is displayed in Figure **??** and shows the purity and ploidy local optima, sorted by final likelihood score after fitting both copy number and allelic fractions.

```
plotAbs(ret, type="overview")
```

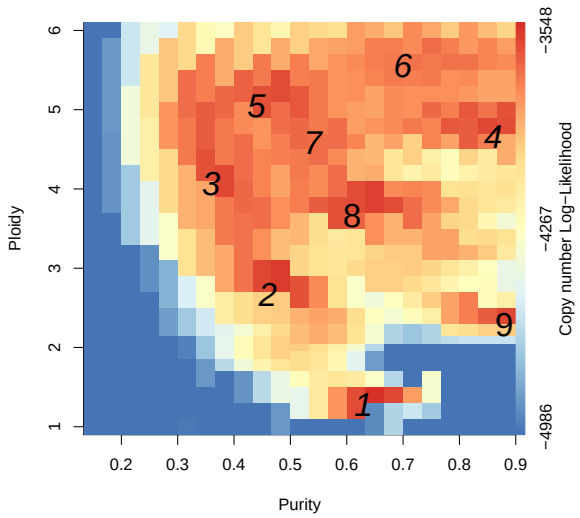

Figure 1: Overview. The colors visualize the copy number fitting score from low (blue) to high (red). The numbers indicate the ranks of the local optima.

We now look at the main plots of the maximum likelihood solution in more detail.

```
plotAbs(ret, 1, type="hist")
```

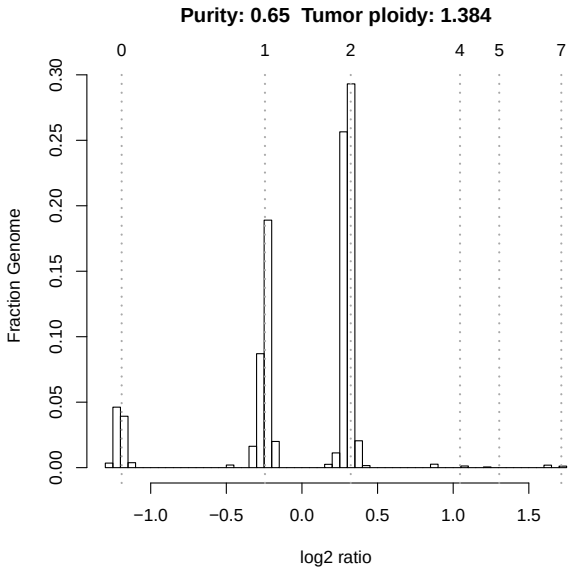

Figure 2: Log-ratio histogram.

Figure **??** displays a histogram of tumor vs. normal copy number log-ratios for the maximum likelihood solution (number 1 in Figure **??**). The height of a bar in this plot is proportional to the fraction of the genome falling into the particular log-ratio copy number range. The vertical dotted lines and numbers visualize the, for the given purity/ploidy combination, expected log-ratios for all integer copy numbers from 0 to 7. It can be seen that most of the log-ratios of the maximum likelihood solution align well to expected values for copy numbers of 0, 1 and 2.

```
plotAbs(ret, 1, type="BAF")
```

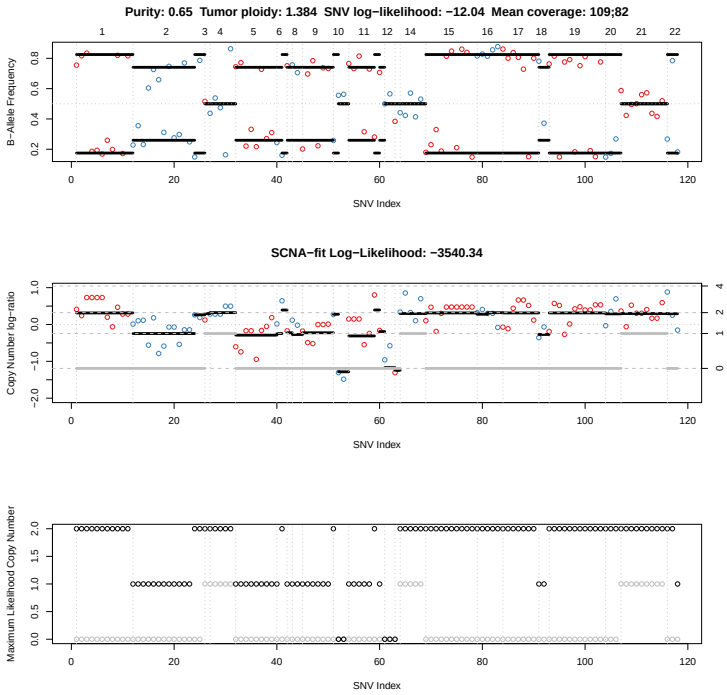

Figure 3: B-allele frequency plot. Each dot is a (predicted) germline SNP. The first panel shows the allelic fractions. The black lines visualize the expected (not the average!) allelic fractions in the segment. These are calculated using the estimated purity and the total and minor segment copy numbers. These are visualized in black and grey, respectively, in the second and third panel. The second panel shows the copy number log-ratios, the third panel the integer copy numbers.

Germline variant data are informative for calculating integer copy number, because unbalanced maternal and paternal chromosome numbers in the tumor portion of the sample lead to unbalanced germline allelic fractions. Figure **??** shows the allelic fractions of predicted germline SNPs. In the middle panel, the corresponding copy number log-ratios are shown. The lower panel displays the calculated integer copy numbers, corrected for purity and ploidy.

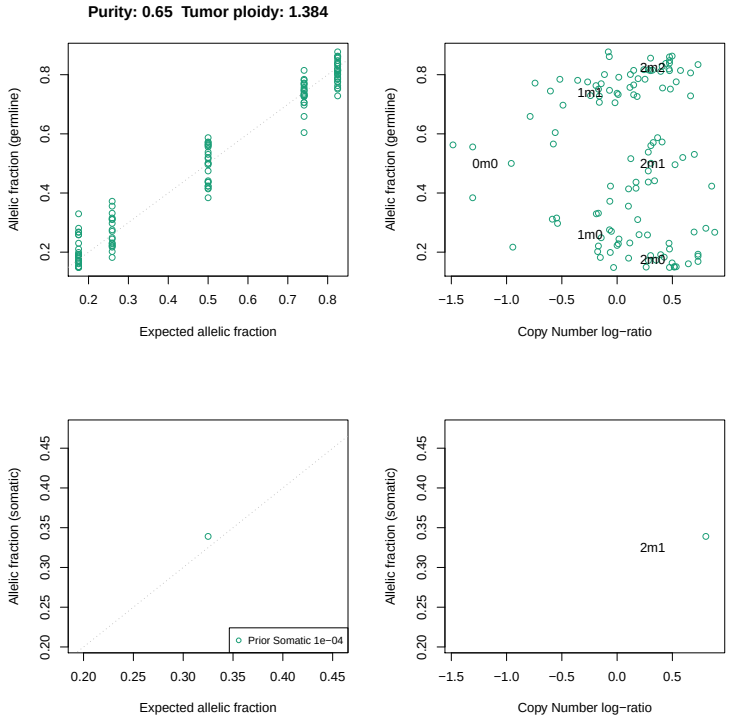

Figure 4: Allele fraction plots. Each dot is again a (predicted) germline SNP. This plot normally also shows somatic mutations in two additional panels. This toy example contains only germline SNPs however.

```
plotAbs(ret, 1, type="AF")
```

Finally, Figure **??** provides a little bit more insight into how well the variants fit the expected values. The left panel shows the correlation of expected and observed allelic fractions. The expected value is determined by the most likely state. High ploidy solutions have a lot of states, so this correlation is expected to be good for high ploidy solutions. The right panel plots the observed allelic fractions against copy number. The labels show the expected values for all called states; 2m1 would be diploid, heterozygous, 2m2 diploid, homozygous.

6 Curation

6.1 Manual

For prediction of SNV status (germline vs. somatic, sub-clonal vs. clonal, homozygous vs. heterozygous), it is important that both purity and ploidy are correct. We provide functionality for curating results:

```
createCurationFile(file.rds)
```

This will generate a CSV file in which the correct purity and ploidy values can be manually entered. It also contains a column "Curated", which should be set to `TRUE`, otherwise the file will be overwritten when re-run.

Then in R, the correct solution (closest to the combination in the CSV file) can be loaded with the `readCurationFile` function:

```
ret <- readCurationFile(file.rds)
```

This function has various handy features, but most importantly it will re-order the local optima so that the curated purity and ploidy combination is ranked first. This means `plotAbs(ret,1,type="hist")` would show the plot for the curated purity/ploidy combination, for example.

The default curation file will list the maximum likelihood solution:

```
read.csv('Sample1_PureCN.csv')
```

```
##   Sampleid Purity   Ploidy Flagged Failed Curated
## 1  Sample1   0.65 1.383545    TRUE  FALSE   FALSE
##                                        Comment
## 1 EXCESSIVE LOSSES;RARE KARYOTYPE;EXCESSIVE LOH
```

PureCN currently only flags samples with warnings, it does not mark any samples as failed. The `Failed` column in the curation file can be used to manually flag samples for exclusion in downstream analyses.

6.2 Automatic

If *PureCN* is mainly used for copy number calling, not for classifying variants, then a deterministic algorithm that does not require manual curation is important and can produce good results for most samples. A default workflow for automatic curation is available in the `autoCurateResults` function. This is a work in progress and implements various heuristics that try to call the correct solution in difficult samples (i.e., noisy or quiet samples). We recommend copying and renaming the function when reproducibility is crucial.

The number of considered local optima is typically high and potentially confusing. The first step in the auto curation is thus to remove low likelihood optima. Internally, this is done with the `bootstrapResults` function. This function bootstraps variants in the provided VCF and re-ranks local optima. Solutions that never rank high are then excluded (Figure **??**). Large-scale copy number artifacts can decrease the likelihood scores of correct solutions, however, potentially resulting in a removal of purity/ploidy estimates closest to the true values.

The next step in the auto curation workflow is rescuing low ranking diploid solutions. True high ploidy solutions typically have a rather uniform copy number distribution, while diploid solutions in general have only a small number of heterozygous losses and single copy gains; a large fraction of the genome would be by definition normal diploid. The `getDiploid` function is internally used to find diploid solutions. Since there should not be any diploid solutions in true high ploidy samples, these diploid solutions are then moved to the top of the ranking. If multiple diploid solutions exist, additional heuristics try to guess the correct one (for example the diploid purity should not be too different from the maximum likelihood purity).

```
ret <- autoCurateResults(ret)

## Bootstrapping VCF to reduce number of solutions.

## Found 0 diploid solutions.

plotAbs(ret, type="overview")
```

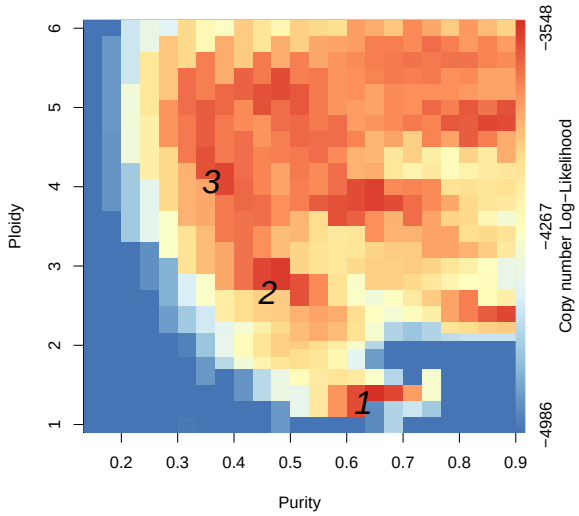

Figure 5: Overview. As in Figure **??**, but with low likelihood solutions automatically removed.

These automatically curated samples can of course be manually curated as well:

```
file.curated.rds <- 'Sample1_PureCN_autocurated.rds'
saveRDS(ret, file=file.curated.rds)
pdf('Sample1_PureCN_autocurated.pdf', width=10, height=12)
```

```
plotAbs(ret, type='all')
dev.off()

## pdf
##   2

createCurationFile(file.curated.rds)
```

7 Custom normalization and segmentation

Copy number normalization and segmentation are crucial for obtaining good purity and ploidy estimates. If you have a well-tested pipeline that produces clean results for your data, you might want (and maybe should) use *PureCN* as add-on to your pipeline. By default, we will use *DNAcopy* [**?**] to segment normalized target-level coverage log-ratios. It is straightforward to replace the default with other methods and the `segmentationCBS` function can serve as an example.

The next section describes how to replace the default segmentation. For the probably more uncommon case that only the coverage normalization is performed by third-party tools, see Section **??**.

7.1 Custom segmentation

It is possible to provide already segmented data, which is especially recommended when matched SNP6 data are available or when third-party segmentation tools are not written in R. Otherwise it is usually however better to customize the default segmentation function, since the algorithm then has access to the raw log-ratio distribution[4]. The expected file format for already segmented copy number data is:

```
ID  chrom   loc.start   loc.end num.mark    seg.mean
Sample1   1   61723   5773942 2681    0.125406444072723
Sample1   1   5774674 5785170 10  -0.756511807441712
```

Since its likelihood model is exon-based, *PureCN* currently still requires an interval file to generate simulated target-level log-ratios from a segmentation file. For simplicity, this interval file is expected either in *GATK DepthOfCoverage* format and provided via the `tumor.coverage.file` argument or via the `gc.gene.file` argument (see Figure **??**). Note that *PureCN* will re-segment the simulated log-ratios using the default `segmentationCBS` function, in particular to identify regions of copy-number neutral LOH and to cluster segments with similar allelic imbalance and log-ratio. The provided interval file should therefore cover all significant copy number alterations[5].

```
retSegmented <- runAbsoluteCN(seg.file=seg.file,
    gc.gene.file=gc.gene.file, vcf.file=vcf.file,
```

[4] If the third-party tool provides target-level log-ratios, then these can be provided via the `log.ratio` argument in addition to `seg.file` though. See also Section **??**.

[5] If this behaviour is not wanted, because maybe the custom function already identifies CNNLOH reliably, `segmentationCBS` can be replaced with a minimal version.

```
    max.candidate.solutions=1, genome="hg19",
    test.purity=seq(0.3,0.7,by=0.05), verbose=FALSE,
    plot.cnv=FALSE)
```

The `max.candidate.solutions` and `test.purity` arguments are set to non-default values to reduce the runtime of this vignette.

```
plotAbs(retSegmented, 1, type="BAF")
```

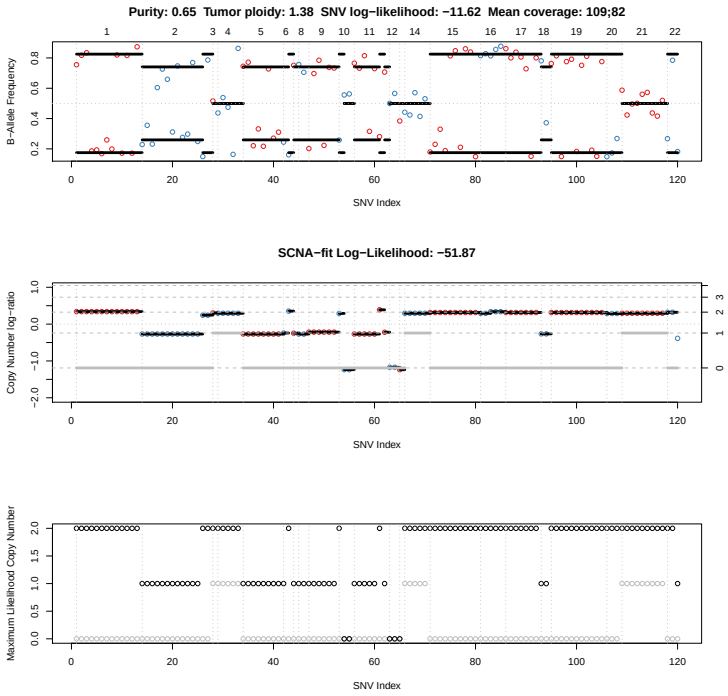

Figure 6: B-allele frequency plot for segmented data. This plot shows the maximum likelihood solution for an example where segmented data are provided instead of coverage data. Note that the middle panel shows no variance in log-ratios, since only segment-level log-ratios are available.

7.2 Custom normalization

If third-party tools such as *GATK4* are used to calculate target-level copy number log-ratios, and *PureCN* should be used for segmentation and purity/ploidy inference only, it is possible to provide these log-ratios:

```
# We still use the log-ratio exactly as normalized by PureCN for this
# example
log.ratio <- calculateLogRatio(readCoverageGatk(normal.coverage.file),
    readCoverageGatk(tumor.coverage.file), verbose=FALSE)
```

```
retLogRatio <- runAbsoluteCN(log.ratio=log.ratio,
    gc.gene.file=gc.gene.file, vcf.file=vcf.file,
    max.candidate.solutions=1, genome="hg19",
    test.purity=seq(0.3,0.7,by=0.05), verbose=FALSE,
    args.filterTargets=list(normalDB=normalDB),
    plot.cnv=FALSE)
```

Again, the `max.candidate.solutions` and `test.purity` arguments are set to non-default values to reduce the runtime of this vignette. It is highly recommended to compare the log-ratios obtained by *PureCN* and the third-party tool, since some pipelines automatically adjust log-ratios for a default purity value. Note that this example uses a pool of normals to filter low quality targets. Interval coordinates are again expected in either a `gc.gene.file` or a `tumor.coverage.file`. If a tumor coverage file is provided, then all targets below the coverage minimum are further excluded.

8 COSMIC annotation

If a matched normal is not available, it is also helpful to provide `runAbsoluteCN` the COSMIC database [**?**] via `cosmic.vcf.file`. While this has limited effect on purity and ploidy estimation due the sparsity of hotspot mutations, it often helps in the manual curation to compare how well high confidence germline (dbSNP) vs. somatic (COSMIC) variants fit a particular purity/ploidy combination.

9 Off-target reads

It is possible to use SNPs in off-target reads in the SNV fitting step by setting the `runAbsoluteCN` argument `remove.off.target.snvs` to `FALSE`. An often better alternative to including all off-target reads is running *MuTect* with interval "padding" (between 50-100bp). Instead of including all off-target SNPs, this will only include SNPs in the flanking regions of targets with `remove.off.target.snvs=FALSE` (see Section **??**).

We recommend a large pool of normals and generating SNP blacklists as described in Sections **??** and **??**.

10 Output

The `plotAbs()` call above will generate the main plots shown in the manuscript. The R data file (file.rds) contains gene-level copy number calls, SNV status and LOH calls. The purity/ploidy combinations are sorted by likelihood and stored in `ret$results`.

```
names(ret)
```

```
## [1] "candidates" "results"    "input"
```

We provide convenient functions to extract information from this data structure and show their usage in the next sections. We recommend using these functions instead of accessing the data directly since data structures might change in future versions.

10.1 Prediction of somatic status and cellular fraction

To understand allelic fractions of particular SNVs, we must know the (i) somatic status, the (ii) tumor purity, the (iii) local copy number, as well as the (iv) number of chromosomes harboring the mutations or SNPs. One of *PureCN* main functions is to find the most likely combination of these four values. We further assign posterior probabilities to all possible combinations or states. Availability of matched normals reduces the search space by already providing somatic status.

The `predictSomatic` function provides access to these probabilities. For predicted somatic mutations, this function also provides cellular fraction estimates, i.e. the fraction of tumor cells with mutation. Fractions significantly below 1 indicate sub-clonality[6]:

[6] This number can be above 1 when the observed allelic fraction is higher than expected for a clonal mutation. This may be due to random sampling, wrong copy number, sub-clonal copy number events, or wrong purity/ploidy estimates.

```
head(predictSomatic(ret), 3)
```

```
##                    chr     start       end SOMATIC.M0   SOMATIC.M1
## chr1114515871xxx chr1 114515871 114515871          0 2.402990e-38
## chr1150044293xxx chr1 150044293 150044293          0 1.222383e-38
## chr1158449835xxx chr1 158449835 158449835          0 7.550793e-62
##                    SOMATIC.M2    SOMATIC.M3 SOMATIC.M4 SOMATIC.M5
## chr1114515871xxx 4.111643e-07 1.143216e-224          0          0
## chr1150044293xxx 4.552325e-10 9.630686e-227          0          0
## chr1158449835xxx 1.127246e-14 1.197342e-228          0          0
##                  SOMATIC.M6 SOMATIC.M7   GERMLINE.M0 GERMLINE.M1
## chr1114515871xxx          0          0  3.852893e-69           0
## chr1150044293xxx          0          0  7.365854e-64           0
## chr1158449835xxx          0          0 3.661565e-105           0
##                  GERMLINE.M2 GERMLINE.M3 GERMLINE.M4 GERMLINE.M5
## chr1114515871xxx   0.9999996           0           0           0
## chr1150044293xxx   1.0000000           0           0           0
## chr1158449835xxx   1.0000000           0           0           0
##                  GERMLINE.M6 GERMLINE.M7 GERMLINE.CONTHIGH
## chr1114515871xxx           0           0      2.458485e-42
## chr1150044293xxx           0           0      9.428729e-21
## chr1158449835xxx           0           0      3.555243e-26
##                  GERMLINE.CONTLOW ML.SOMATIC ML.M ML.C
## chr1114515871xxx    5.657573e-288      FALSE    2    2
## chr1150044293xxx    1.122262e-242      FALSE    2    2
```

```
## chr1158449835xxx     0.000000e+00      FALSE    2    2
##                  ML.M.Segment ML.AR       AR AR.ADJUSTED
## chr1114515871xxx            0 0.825 0.755183   0.7760674
## chr1150044293xxx            0 0.825 0.817078   0.8396741
## chr1158449835xxx            0 0.825 0.834266   0.8573374
##                  MAPPING.BIAS CN.Subclonal Log.Ratio Prior.Somatic
## chr1114515871xxx    0.9730894        FALSE 0.4110604   9.90099e-05
## chr1150044293xxx    0.9730894        FALSE 0.2418054   9.90099e-05
## chr1158449835xxx    0.9730894        FALSE 0.7319846   9.90099e-05
##                  Prior.Contamination ML.LOH gene.symbol
## chr1114515871xxx                0.01   TRUE       HIPK1
## chr1150044293xxx                0.01   TRUE       VPS45
## chr1158449835xxx                0.01   TRUE      OR10R2
##                  Cellfraction
## chr1114515871xxx           NA
## chr1150044293xxx           NA
## chr1158449835xxx           NA
```

M0 to M7 are multiplicity values, i.e. the number of chromosomes harboring the mutation (e.g. 1 heterozygous, 2 homozygous if copy number C is 2). Columns with the ML prefix indicate maximum likelihood estimates, e.g. ML.AR is the expected allelic ratio of the most likely state, AR is the observed allelic ratio as provided in the VCF file. GERMLINE.CONTHIGH and GERMLINE.CONTLOW are the two contamination states. The former are homozygous germline SNPs that were not filtered out because reference alleles from another individual were sequenced, resulting in allelic fractions smaller than 1. The latter are non-reference alleles only present in the contamination.

To annotate the input VCF file with these values:

```
vcf <- predictSomatic(ret, return.vcf=TRUE)
writeVcf(vcf, file="Sample1_PureCN.vcf")
```

Note that the posterior probabilities assume that the purity and ploidy combination is correct. Before classifying variants, it is thus important to manually curate samples. Further note that small inaccuracies in purity can decrease the classification performance significantly, and we currently recommend the `post.optimize` option when variant classification is important.

10.2 Amplifications and deletions

To call amplifications, we recommend using a cutoff of 6 for focal amplifications and a cutoff of 7 otherwise. For homozygous deletions, a cutoff of 0.5 is useful to allow some heterogeneity in copy number.

For samples that failed *PureCN* calling we recommended using common log-ratio cutoffs to call amplifications, for example 0.9.

This strategy is implemented in the `callAlterations` function:

```
gene.calls <- callAlterations(ret)
head(gene.calls)
```

```
##        chr     start       end C seg.mean seg.id number.exons
## EIF2A chr3 150264590 150301699 7   1.6343      5           14
## AADAC chr3 151531951 151545961 7   1.6343      5            5
## FGF5  chr4  81187979  81207827 0  -1.2008      9            3
## SPP1  chr4  88898048  88904049 0  -1.2008      9            7
## PPM1K chr4  89183747  89199736 0  -1.2008      9            6
## MMRN1 chr4  90816123  90874570 0  -1.2008      9            8
##       gene.mean   gene.min   gene.max focal          type
## EIF2A  1.782366  0.5916938  2.3573686  TRUE AMPLIFICATION
## AADAC  1.060134  0.6476273  1.3938954  TRUE AMPLIFICATION
## FGF5  -1.017110 -1.1767409 -0.8132705 FALSE      DELETION
## SPP1  -1.271431 -1.5553359 -0.6862161 FALSE      DELETION
## PPM1K -1.141944 -1.7978892 -0.4340134 FALSE      DELETION
## MMRN1 -1.194234 -1.9457390 -0.7587568 FALSE      DELETION
##       num.snps.segment loh
## EIF2A                0  NA
## AADAC                0  NA
## FGF5                 0  NA
## SPP1                 0  NA
## PPM1K                0  NA
## MMRN1                0  NA
```

It is also often useful to filter the list further by known biology, for example to exclude non-focal amplifications of tumor suppressor genes. The Sanger Cancer Gene Census [**?**] for example provides such a list.

10.3 Find genomic regions in LOH

The `gene.calls` data.frame described above provides gene-level LOH information. To find the corresponding genomic regions in LOH, we can use the `callLOH` function:

```
loh <- callLOH(ret)
head(loh)
```

```
##    chr     start       end arm C M                      type
## 1 chr1 114515871 121535434   p 2 0 WHOLE ARM COPY-NEUTRAL LOH
## 2 chr1 124535434 247419499   q 2 0 WHOLE ARM COPY-NEUTRAL LOH
## 3 chr2  10262881  92326171   p 1 0             WHOLE ARM LOH
## 4 chr2  95326171 231775198   q 1 0             WHOLE ARM LOH
## 5 chr2 236403412 239039169   q 2 0          COPY-NEUTRAL LOH
## 6 chr3  11888043  90504854   p 2 1
```

11 Power to detect somatic mutations

As final quality control step, we can test if coverage and tumor purity are sufficent to detect mono-clonal or even sub-clonal somatic mutations. We strictly follow the power calculation by Carter et al. [**?**].

The following Figure **??** shows the power to detect mono-clonal somatic mutations as a function of tumor purity and sequencing coverage (reproduced from [**?**]):

```
purity <- c(0.1,0.15,0.2,0.25,0.4,0.6,1)
coverage <- seq(5,35,1)
power <- lapply(purity, function(p) sapply(coverage, function(cv)
    calculatePowerDetectSomatic(coverage=cv, purity=p, ploidy=2,
    verbose=FALSE)$power))

# Figure S7b in Carter et al.
plot(coverage, power[[1]], col=1, xlab="Sequence coverage",
    ylab="Detection power", ylim=c(0,1), type="l")

for (i in 2:length(power)) lines(coverage, power[[i]], col=i)
abline(h=0.8, lty=2, col="grey")
legend("bottomright", legend=paste("Purity", purity),
    fill=seq_along(purity))
```

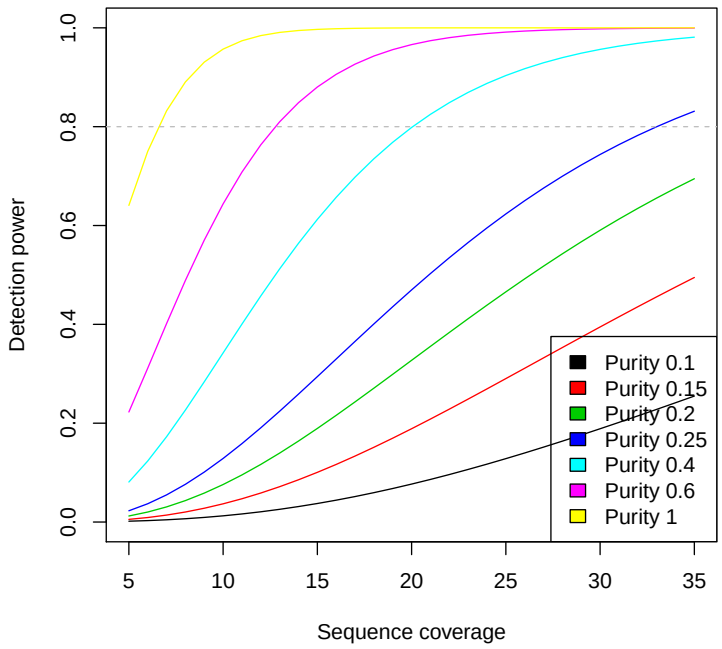

Figure 7: Power to detect mono-clonal somatic mutations. Reproduced from [**?**].

Figure **??** then shows the same plot for sub-clonal mutations present in 20% of all tumor cells:

```
coverage <- seq(5,350,1)
power <- lapply(purity, function(p) sapply(coverage, function(cv)
    calculatePowerDetectSomatic(coverage=cv, purity=p, ploidy=2,
    cell.fraction=0.2, verbose=FALSE)$power))
plot(coverage, power[[1]], col=1, xlab="Sequence coverage",
    ylab="Detection power", ylim=c(0,1), type="l")

for (i in 2:length(power)) lines(coverage, power[[i]], col=i)
abline(h=0.8, lty=2, col="grey")
legend("bottomright", legend=paste("Purity", purity),
    fill=seq_along(purity))
```

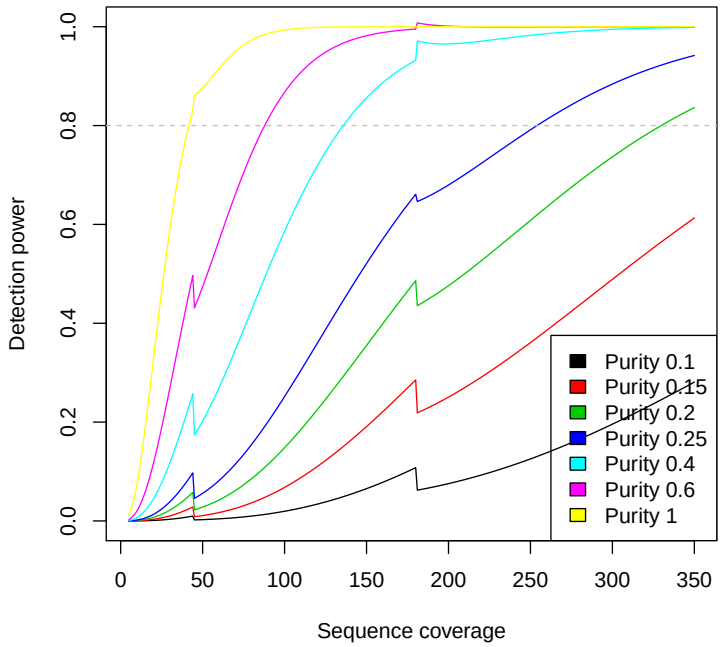

Figure 8: **Power to detect sub-clonal somatic mutations present in 20% of all tumor cells.** Reproduced from [?].

12 Command line scripts

12.1 Coverage

We provide a basic template for GC-normalizing BAM or *GATK DepthOfCoverage* files from the command line. For example:

```
# From a BAM file
Rscript Coverage.R --outdir ~/tmp/ --bam ex1.bam \
     --gcgene ex1_intervals.txt

# From a GATK DepthOfCoverage file
Rscript Coverage.R --outdir ~/tmp/ --gatkcoverage example_tumor.txt \
    --gcgene  example_gc.gene.file.txt
```

Table 1: Coverage command line script.

Argument name	Corresponding PureCN argument	PureCN function
–help -h		
–outdir -o		
–bam -b	bam.file	`calculateBamCoverageByInterval`
–gatkcoverage -g	coverage.file	`correctCoverageBias`
–gcgene -c	gc.gene.file	`correctCoverageBias`

12.2 PureCN

PureCN.R is an example script to run PureCN with basic parameters:

```
# From GC-normalized coverage data
Rscript PureCN.R --outdir ~/tmp/ --tumor example_tumor.txt \
    --normal example_normal.txt --vcf example_vcf.vcf -i Sample1 \
     --genome hg19 --gcgene  example_gc.gene.file.txt

# From already segmented data
Rscript PureCN.R --outdir ~/tmp/ --segfile example_seg.txt \
    --vcf example_vcf.vcf -i Sample1 --genome hg19 \
    --gcgene  example_gc.gene.file.txt
```

These R scripts can be found in the extdata directory of the installed package.

Table 2: PureCN command line script.

Argument name	Corresponding PureCN argument	PureCN function
–help -h		
–outdir -o		
–normal -n	normal.coverage.file	`runAbsoluteCN`
–tumor -t	tumor.coverage.file	`runAbsoluteCN`
–vcf -v	vcf.file	`runAbsoluteCN`
–genome -g	genome	`runAbsoluteCN`
–gcgene -c	gc.gene.file	`runAbsoluteCN`
–segfile -f	seg.file	`runAbsoluteCN`
–snpblacklist -s	snp.blacklist	`filterVcfBasic`
–statsfile -a	stats.file	`filterVcfMuTect`
–targetweightfile -e	target.weight.file	`segmentationCBS`
–normaldb -d	normalDB (serialized with `saveRDS`)	`findBestNormal`, `filterTargets`
–sampleid -i	sampleid	`runAbsoluteCN`

13 Limitations

PureCN currently assumes a completely diploid normal genome. For human samples, it tries to detect sex by calculating the coverage ratio of chromosomes X and Y and will then remove sex chromosomes in male samples[7]. For non-human samples, the user needs to manually remove all non-diploid chromosomes from the coverage data and specify `sex='diploid'` in the *PureCN* call.

[7] Loss of Y chromosome (LOY) can result in wrong female calls, especially in high purity samples or if LOY is in both tumor and contaminating normal cells. The software throws a warning in this case when germline SNPs are provided.

While *PureCN* supports and models sub-clonal somatic copy number alterations, it currently assumes that the majority of alterations are mono-clonal. For most clinical samples, this is reasonable, but very heterogeneous samples are likely not possible to call without manual curation (but please note that other algorithms that model poly-genomic tumors do not necessarily have a higher call rate, since they tend to overfit noisy samples). Due to the lack of signal, manual curation is also recommended in low purity samples or very quiet genomes.

In the absence of matched normals, the software currently requires some normal contamination to infer germline genotypes. Since cell lines are rarely matched, *PureCN* will likely not work well with cell line data.

Finally, the software currently does not officially support VCF files containing indels. Support for VCFs generated by *MuTect 2* that include both single nucleotide variants (SNVs) and indels is planned for Bioconductor 3.5. Experimental support for *MuTect 2* VCFs generated in tumor-only mode is available now.

14 FAQ

If the ploidy is frequently too high, please check:

- Does the log-ratio histogram (Figure **??**) look noisy? If yes, then

- Is the coverage sufficient? Tumor coverages below 80X can be difficult, especially in low purity samples. Normal coverages below 50X might result in high variance of log-ratios. See Section **??** for finding a good normal sample for log-ratio calculation.
- Is the coverage data of both tumor and normal GC-normalized? If not, see `correctCoverageBias`.
- Is the quality of both tumor and normal sufficient? A high AT or GC-dropout might result in high variance of log-ratios. Challenging FFPE samples also might need parameter tuning of the segmentation function. See `segmentationCBS`. A high expected tumor purity allows more aggressive segmentation parameters, such as `prune.hclust.h=0.2` or higher.
- Was the correct target interval file used (genome version and capture kit, see Section **??**)? If unsure, ask the help desk of your sequencing center.
- Were the normal samples run with the same assay and pipeline?
- Did you provide `runAbsoluteCN` all the recommended files as described in Section **??**?
- For whole-genome data, you most likely get better results using a specialized third-party segmentation method as described in section **??**, since our default is optimized for targeted sequencing.

- Otherwise, if log-ratio peaks are clean as in Figure **??**:
  - Was MuTect run without a matched normal? If yes, then make sure to provide a SNP blacklist as described in Sections **??** and **??**.
  - A high fraction of sub-clonal copy-number alterations might also result in a low ranking of correct low ploidy solutions.

Will PureCN work with my data?

- *PureCN* was designed for medium-sized (>2-3Mb) targeted panels. The more data, the better, best results are typically achieved in whole-exome data.
- The same is obviously true for coverage. Coverages below 80X are difficult unless purities are high and coverages are even.
- The number of heterozygous SNPs is also important. Copy number probes enriched in SNPs are therefore very helpful.
- While *PureCN* supports to some degree uneven tiling of targets, the more evenly distributed the better. Large gaps are problematic and might require third-party segmentation tools that support off-target reads (Section **??**).
- Whole-genome data is not officially supported and specialized tools will likely provide better results. Third-party segmentation tools designed for this data type would be again required.

- *PureCN* also needs process-matched normal samples, again, the more the better.

If you have trouble generating input data PureCN accepts, please check:

- For problems related to generating valid coverage data, either consult the *GATK* manual for the *DepthOfCoverage* tool or Section **??** for the equivalent function in *PureCN*.
- Currently only VCF files generated by *MuTect 1* are officially supported and well tested. A minimal example *MuTect* call would be:

```
$JAVA -Xmx6g -jar $MUTECT \
--analysis_type MuTect -R $REFERENCE \
--dbsnp $DBSNP_VCF \
--cosmic $COSMIC_VCF \
-I:normal $BAM_NORMAL \
-I:tumor $BAM_TUMOR  \
-o $OUT/${ID}_bwa_mutect_stats.txt \
-vcf $OUT/${ID}_bwa_mutect.vcf
```

  The default output file is the stats or call-stats file; this can be provided in addition to the required VCF file via `args.filterVcf` in `runAbsoluteCN`. If provided, it may help *PureCN* filter artifacts. This requires *MuTect* in version 1.1.7. Note that this *MuTect* call will keep all off-target calls. We recommend running *MuTect* with interval file, but include flanking regions:

```
$JAVA -Xmx6g -jar $MUTECT \
--analysis_type MuTect -R $REFERENCE \
--dbsnp $DBSNP_VCF \
--cosmic $COSMIC_VCF \
--interval_padding 75 \
--L $INTERVAL_FILE \
-I:normal $BAM_NORMAL \
-I:tumor $BAM_TUMOR  \
-o $OUT/${ID}_bwa_mutect_stats.txt \
-vcf $OUT/${ID}_bwa_mutect.vcf
```

  We highly recommend finding good values for the `interval_padding` argument for each assay. A good cutoff will maximize the number of heterozygous SNPs and keep only an acceptable number of lower quality calls. To include SNPs in the flanking regions, it is necessary to set `remove.off.target.snvs=FALSE`. Future versions will automatically keep these variants.

Questions related to manual curation. *PureCN*, like most other related tools, essentially finds the most simple explanation of the data. There are three major problems with this approach:

- First, hybrid capture data can be noisy and the algorithm must distinguish signal from noise; if the algorithm mistakes noise for signal, then this often results in wrong high ploidy calls (see Sections **??** and **??**). If all steps in this vignette were followed, then *PureCN* should do a good job in ignoring common artifacts. Noisy samples thus often have outlier ploidy values and are often automatically flagged by *PureCN*. The correct solution is in most of these cases ranked second or third.
- The second problem is that signal can be sparse, i.e. when the tumor purity is very low or when there are only few somatic events. Manual curation is often easy in the latter case. For example when small losses are called as homozygous, but corresponding germline allele-frequencies are unbalanced (a complete loss would result in balanced germline allele frequencies, since only normal DNA is left). Future versions might improve calling in these cases by underweighting uninformative genomic regions.
- The third problem is that tumor evolution is fast and complex and very difficult to incorporate into general likelihood models. Sometimes multiple solutions explain the data equally well, but one solution is then often clearly more consistent with known biology, for example LOH in tumor suppressor genes such as *TP53*. A basic understanding of both the algorithm and the tumor biology of the particular cancer type are thus important for curation. Fortunately, in most cancer types, such ambiguity is rather rare.

If all or most of the samples are flagged as:

Noisy segmentation: The default of 200 for `max.segments` is calibrated for high quality and high coverage whole exome data. For whole genome data or lower coverage data, this value needs to be re-calibrated. In case the copy number data looks indeed noisy, please see the first FAQ.

High AT/GC dropout: If the data is GC-normalized, then there might be issues with either the target intervals or the provided GC content. Please double check that all files are correct and that all the coverage files are GC-normalized (Section **??**).

Annotating intervals with gene symbols

A simple function that annotates intervals with the **first** overlapping interval from the RefSeq gene track:

```
tblGenes <- NULL
if (downloadFromUCSC) {
    library(rtracklayer)
    mySession <- browserSession("UCSC")
    genome(mySession) <- "hg19"
    tblGenes <- getTable( ucscTableQuery(mySession,
        track="RefSeq Genes", table="refGene"))
```

```
}

.annotateIntervals <- function(gc.gene.file, tblGenes,
    output.file = NULL) {
    grGenes <- GRanges(seqnames=tblGenes$chrom,
        IRanges(start=tblGenes$cdsStart, end=tblGenes$cdsEnd))
    gc <- read.delim(gc.gene.file, as.is=TRUE)
    # misuse this function to convert interval string into data.frame
    gc.data <- readCoverageGatk(gc.gene.file)
    grGC <- GRanges(seqnames=gc.data$chr,
        IRanges(start=gc.data$probe_start, end=gc.data$probe_end))
    ov <- findOverlaps(grGC, grGenes, select="first")
    gc$Gene <- tblGenes$name2[ov]
    if (!is.null(output.file)) {
        write.table(gc, file = output.file, row.names = FALSE,
            quote = FALSE, sep = "\t")
    }
    invisible(gc)
}

if (!is.null(tblGenes)) .annotateIntervals(gc.gene.file, tblGenes)
```

Session Info

- R version 3.3.1 (2016-06-21), `x86_64-apple-darwin13.4.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: Biobase 2.34.0, BiocGenerics 0.20.0, Biostrings 2.42.0, DNAcopy 1.48.0, GenomeInfoDb 1.10.0, GenomicRanges 1.26.1, IRanges 2.8.0, PureCN 1.2.3, Rsamtools 1.26.1, S4Vectors 0.12.0, SummarizedExperiment 1.4.0, VariantAnnotation 1.20.0, XVector 0.14.0
- Loaded via a namespace (and not attached): AnnotationDbi 1.36.0, BSgenome 1.42.0, BiocParallel 1.8.0, BiocStyle 2.2.0, DBI 0.5-1, GenomicAlignments 1.10.0, GenomicFeatures 1.26.0, Matrix 1.2-7.1, RColorBrewer 1.1-2, RCurl 1.95-4.8, RSQLite 1.0.0, XML 3.98-1.4, biomaRt 2.30.0, bitops 1.0-6, chron 2.3-47, data.table 1.9.6, evaluate 0.10, formatR 1.4, grid 3.3.1, highr 0.6, knitr 1.14, lattice 0.20-34, magrittr 1.5, rtracklayer 1.34.0, stringi 1.1.2, stringr 1.1.0, tools 3.3.1, zlibbioc 1.20.0