cn.mops - Mixture of Poissons for CNV detection in NGS data

Günter Klambauer

Institute of Bioinformatics, Johannes Kepler University Linz
Altenberger Str. 69, 4040 Linz, Austria
cn.mops@bioinf.jku.at

Version 1.55.1, July 23, 2025

Institute of Bioinformatics
Johannes Kepler University Linz
A-4040 Linz, Austria

Tel. +43 732 2468 8880
Fax +43 732 2468 9511
http://www.bioinf.jku.at

Contents

1   Introduction

The `cn.mops` package is part of the Bioconductor (`http://www.bioconductor.org`) project. The package allows to detect copy number variations (CNVs) from next generation sequencing (NGS) data sets based on a generative model. Please visit `http://www.bioinf.jku.at/software/cnmops/cnmops.html` for additional information.

To avoid the false discoveries induced by read count variations along the chromosome or across samples, we propose a "Mixture Of PoissonS model for CNV detection" (cn.MOPS). The cn.MOPS model is not affected by read count variations along the chromosome, because at each DNA position a local model is constructed. Read count variations across samples are decomposed by the cn.MOPS model into integer copy numbers and noise by its mixture components and Poisson distributions, respectively. In contrast to existing methods, cn.MOPS model's posterior provides integer copy numbers together with their uncertainty. Model selection in a Bayesian framework is based on maximizing the posterior given the samples by an expectation maximization (EM) algorithm. The model incorporates the linear dependency between average read counts in a DNA segment and its copy number. Most importantly, a Dirichlet prior on the mixture components prefers constant copy number 2 for all samples. The more the data drives the posterior away from the Dirichlet prior corresponding to copy number two, the more likely the data is caused by a CNV, and, the higher is the informative/non-informative (I/NI) call. cn.MOPS detects a CNV in the DNA of an individual as a segment with high I/NI calls. I/NI call based CNV detection guarantees a low false discovery rate (FDR) because wrong detections are less likely for high I/NI calls. We assume that the genome is partitioned into segments in which reads are counted but which need not be of constant length throughout the genome. For each of such an segment we build a model. We consider the read counts x at a certain segment of the genome, for which we construct a model across samples. The model incorporates both read count variations due to technical or biological noise and variations stemming from copy number variations. For further information regarding the algorithm and its assessment see the cn.MOPS homepage at `http://www.bioinf.jku.at/software/cnmops/cnmops.html`.

2   Getting started and quick start

To load the package, enter the following in your R session:

```
> library(cn.mops)
```

The whole pipeline will only take a few steps, if BAM files are available (for read count matrices directly go to step 2):

1. Getting the input data from BAM files (also see Section 3.2 and Section 3).

   ```
   > BAMFiles <- list.files(pattern=".bam$")
   > bamDataRanges <- getReadCountsFromBAM(BAMFiles)
   ```

2. Running the algorithm (also see Section 4.2).

   ```
   > res <- cn.mops(bamDataRanges)
   ```

3. Visualization of the detected CNV regions. For more information about the result objects and visualization see Section 4.2.

```
> plot(res,which=1)
```

Normalized Read Counts

Read Count Ratios

chrA: 48325001 − 48900000

chrA: 48325001 − 48900000

Local Assessments

CNV Call

chrA: 48325001 − 48900000

chrA: 48325001 − 48900000

3   Input of `cn.mops`: BAM files, GRanges objects, or numeric matrices

3.1   Read count matrices as input

`cn.mops` does not require the data samples to be of any specific kind or structure. `cn.mops` only requires a *read count matrix*, i.e., given N data samples and m genomic segments, this is an $m \times N$ real- or integer-valued matrix $\mathbf{X}$, in which an entry x_{ij} corresponds to the read count of sample j in the i-th segment. E.g. in the following read count matrix sample three has 17 reads in the second segment: $x_{23} = 71$.

$$
\mathbf{X} = \begin{array}{r}
 \\
\text{Segment 1} \\
\text{Segment 2} \\
\text{Segment 3} \\
\text{Segment 4} \\
\text{Segment 5} \\
\text{Segment 6}
\end{array}
\begin{array}{cccc}
\text{Sample 1} & \text{Sample 2} & \text{Sample 3} & \text{Sample 4} \\
\left(\begin{array}{cccc}
88 & 82 & 79 & 101 \\
83 & 78 & 71 & 99 \\
43 & 50 & 55 & 37 \\
47 & 58 & 48 & 42 \\
73 & 86 & 95 & 91 \\
92 & 90 & 80 & 71
\end{array} \right)
\end{array}
$$

`cn.mops` can handle numeric and integer matrices or `GRanges` objects, in which the read counts are stored as `values` of the object.

3.2   BAM files as input

The most widely used file format for aligned short reads is the Sequence Alignment Map (SAM) format or in the compressed form the Binary Alignment Map (BAM). We provide a simple function that makes use of the `Rsamtools` package to obtain the alignment positions of reads. The result object of the function can directly be used as input for `cn.mops`. The author can provide functions for input formats other than BAM upon request: *cn.mops@bioinf.jku.at* .

```
> BAMFiles <- list.files(system.file("extdata", package="cn.mops"),pattern=".bam$",
+               full.names=TRUE)
> bamDataRanges <- getReadCountsFromBAM(BAMFiles,
+               sampleNames=paste("Sample",1:3))
```

In `bamDataRanges` you have now stored the genomic segments (left of the |'s) and the read counts (right of the |'s):

```
> (bamDataRanges)
```

```
GRanges object with 2522 ranges and 3 metadata columns:
         seqnames              ranges strand |  Sample 1  Sample 2  Sample 3
            <Rle>           <IRanges>  <Rle> | <integer> <integer> <integer>
    [1]        20             1-25000      * |         0         0         0
    [2]        20         25001-50000      * |         0         0         0
    [3]        20         50001-75000      * |       762       848       784
    [4]        20        75001-100000      * |         0         0         0
    [5]        20       100001-125000      * |         0         0         0
    ...       ...                 ...    ... .       ...       ...       ...
 [2518]        20 62925001-62950000      * |         0         0         0
 [2519]        20 62950001-62975000      * |         0         0         0
 [2520]        20 62975001-63000000      * |         0         0         0
 [2521]        20 63000001-63025000      * |         0         0         0
 [2522]        20 63025001-63025520      * |         0         0         0
 -------
 seqinfo: 1 sequence from an unspecified genome
```

4 Copy number estimation with `cn.mops`

To get a first impression, we use a data set, in which CNVs have been artificially implanted. The simulated data set was generated using distributions of read counts as they appear in real sequencing experiments. CNVs were implanted under the assumption that the expected read count is linear dependent on the copy number. For example in a certain genomic we expect λ reads for copy number 2, then we expect 2λ reads for copy number 4. The linear relationship was confirmed in different studies, like Alkan *et al.* (2009), Chiang *et al.* (2008) and Sathirapongsasuti *et al.* (2011).

4.1 Running `cn.mops`

The read counts are stored in the objects X and XRanges, which are the two basic input types that `cn.mops` allows:

```
> data(cn.mops)
> ls()
```

```
[1] "BAMFiles"       "CNVRanges"      "X"             "XRanges"
[5] "bamDataRanges"  "cn.mopsVersion" "exomeCounts"   "resCNMOPS"
```

The same data is stored in a `GRanges` object, in which we see the genomic coordinates, as well as the read counts (values):

```
> head(XRanges[,1:3])
```

```
GRanges object with 6 ranges and 3 metadata columns:
       seqnames          ranges strand |         S_1        S_2        S_3
          <Rle>       <IRanges>  <Rle> | <integer> <integer> <integer>
  [1]     chrA         1-25000      * |         102         93        109
  [2]     chrA     25001-50000      * |         118         99        100
  [3]     chrA     50001-75000      * |          85         81         82
  [4]     chrA    75001-100000      * |          87        116        106
  [5]     chrA   100001-125000      * |          87         68         89
  [6]     chrA   125001-150000      * |          87         91         91
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

We are now ready to run `cn.mops` on the `GRanges` object:

```
> resCNMOPS <- cn.mops(XRanges)
```

To calculate integer copy number use the command `calcIntegerCopyNumbers`:

```
> resCNMOPS <- calcIntegerCopyNumbers(resCNMOPS)
```

Alternatively, it is possible to use an integer matrix, in which the genomic coordinates can be stored as `rownames` and the entries are the read counts. For example the data from above represented by an integer matrix X:

```
> head(X[,1:3])
```

```
                    S_1 S_2 S_3
Chr_A_1_25000       102  93 109
Chr_A_25001_50000   118  99 100
Chr_A_50001_75000    85  81  82
Chr_A_75001_100000   87 116 106
Chr_A_100001_125000  87  68  89
Chr_A_125001_150000  87  91  91
```

We are now ready to run `cn.mops` on the integer matrix:

```
> resCNMOPSX <- cn.mops(X)
```

To calculate integer copy number use the command `calcIntegerCopyNumbers`:

```
> resCNMOPSX <- calcIntegerCopyNumbers(resCNMOPSX)
```

Note that the two results `resCNMOPS` and `resCNMOPSRanges` identify the same CNVs:

```
> all(individualCall(resCNMOPSX)==individualCall(resCNMOPS))
```

4.2   The result object

To get a summary of the CNV detection result, just enter the name of the object (which implicitly calls `show`):

```
> (resCNMOPS)
```

The CNVs per individual are stored in the slot `cnvs`:

```
> cnvs(resCNMOPS)[1:5]
```

```
GRanges object with 5 ranges and 4 metadata columns:
      seqnames              ranges strand |  sampleName     median
         <Rle>           <IRanges>  <Rle> | <character>  <numeric>
  [1]     chrA    1775001-1850000      * |         S_1   -1.00000
  [2]     chrA   53300001-53400000      * |         S_1   -5.32193
  [3]     chrA 113175001-113325000      * |         S_1   -0.99991
  [4]     chrA   48575001-48650000      * |         S_2   -5.32193
  [5]     chrA   70625001-70750000      * |         S_2   -5.32193
           mean          CN
```

```
      <numeric> <character>
[1]  -0.999999          CN1
[2]  -5.321928          CN0
[3]  -0.999682          CN1
[4]  -5.321928          CN0
[5]  -5.321928          CN0
-------
seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Segments, in which individual CNVs accumulate, are called CNV regions and can be accessed by cnvr:

```
> cnvr(resCNMOPS)[1,1:5]
```

```
GRanges object with 1 range and 5 metadata columns:
      seqnames              ranges strand |          S_1          S_2
         <Rle>           <IRanges>  <Rle> |  <character>  <character>
  [1]      chrA 1775001-1850000        * |          CN1          CN2
            S_3          S_4          S_5
    <character>  <character>  <character>
  [1]          CN2          CN2          CN2
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

We now want to check, whether cn.mops found the implanted CNVs. We have stored the implanted CNVs (see beginning of Section) in the object CNVRanges.

```
> (CNVRanges[15,1:5])
```

```
GRanges object with 1 range and 5 metadata columns:
      seqnames              ranges strand |          S_1          S_2          S_3
         <Rle>           <IRanges>  <Rle> |    <integer>    <integer>    <integer>
  [1]      chrA 1751017-1850656        * |            1            2            2
            S_4          S_5
      <integer>    <integer>
  [1]            2            2
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Next we identify overlaps between CNVs that were detected by cn.mops and CNVs that were implanted. Towards this end we use the functions of the GenomicRanges package.

```
> ranges(cnvr(resCNMOPS))[1:2]
```

```
IRanges object with 2 ranges and 0 metadata columns:
          start         end       width
      <integer>   <integer>   <integer>
  [1]   1775001     1850000       75000
  [2]   5525001     5600000       75000
```

```
> ranges(cnvr(resCNMOPS)) %over% ranges(CNVRanges)
```

```
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[15] TRUE TRUE TRUE TRUE TRUE
```

The detected CNV regions all overlap with the known CNV regions contained in CNVRanges.

The function cn.mops creates an instance of the S4 class CNVDetectionResult that is defined by the present package. To get detailed information on which data are stored in such objects, enter

```
> help(CNVDetectionResult)
```

5   Visualization of the result

5.1   Chromosome plots

cn.mops allows for plotting the detected segments of an individual at one chromosome by a plot similar to the ones produced by DNAcopy:

```
> segplot(resCNMOPS,sampleIdx=13)
```

S_13

Chromosome chrA

Figure 1: The x-axis represents the genomic position and on the y-axis we see the log ratio of the read counts (green) and the copy number call of each segment (red).

5.2   CNV region plots

cn.mops allows for plotting the detected CNV regions:

```
> plot(resCNMOPS,which=1)
```

Figure 2: The x-axis represents the genomic position and on the y-axis we see the read counts (left), the call of the local model (middle) and the CNV call produced by the segmentation algorithm. Blue lines mark samples having a copy number loss.

6   Exome sequencing data

To apply cn.mops to exome sequencing data requires a different preprocessing, since constant windows spanning the whole genome are not appropiate. The initial segments in which the reads are counted should be chosen as the regions of the baits, targets or exons. The read count matrix can now be generated by using the function getSegmentReadCountsFromBAM that requires the genomic coordinates of the predefined segments as GRanges object. The resulting read count

matrix can directly be used as input for cn.mops. A possible processing script could look like the following:

```
> library(cn.mops)
> BAMFiles <- list.files(pattern=".bam$")
> segments <- read.table("targetRegions.bed",sep="\t",as.is=TRUE)
> gr <- GRanges(segments[,1],IRanges(segments[,2],segments[,3]))
> X <- getSegmentReadCountsFromBAM(BAMFiles,GR=gr)
> resCNMOPS <- exomecn.mops(X)
> resCNMOPS <- calcIntegerCopyNumbers(resCNMOPS)
```

We included an exome sequencing data set in this package. It is stored in exomeCounts.

```
> resultExomeData <- exomecn.mops(exomeCounts)
> resultExomeData  <- calcIntegerCopyNumbers(resultExomeData )
```
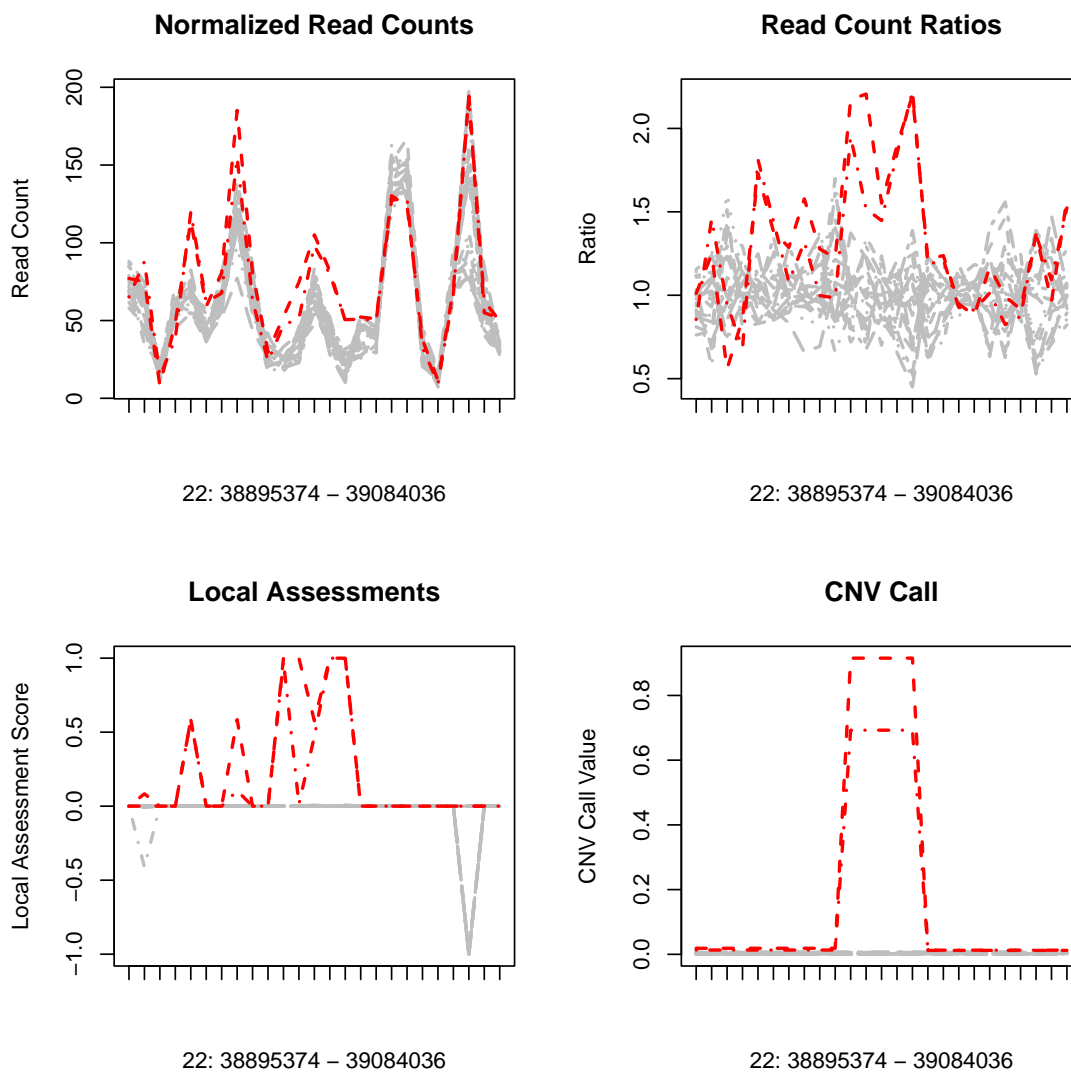
```
> plot(resultExomeData,which=5)
```

Figure 3: ExomeSeq data results.

Possible issues and notes A problem can occur, if the names of the reference sequences, e.s. chromosomes, are inconsistent between the bed file and the bam file. For example "chr1", "chr2",...,"chrX","chrY" and "1","2",...,"X","Y".This can easily be solved by replacing the seqlevels of the GRanges object:

```
> #the following removes the "chr" from reference sequence names
> library(Seqinfo)
> seqlevels(gr) <- gsub("chr","",seqlevels(gr))
```

Results can also be improved if you extend your target regions by a small amount of bases to the left and to the right (in the following case it is 30bp):

```
> gr <- GRanges(segments[,1],IRanges(segments[,2]-30,segments[,3]+30))
> gr <- reduce(gr)
```

7   Cases vs. Control or Tumor vs. Normal

For detection of CNVs in a setting in which the normal state is known, the function referencecn.mops
can be applied. It implements the cn.MOPS algorithm adjusted to this setting. For tumor samples
very high copy numbers can be present – the maximum copy number with the default setting is 8
– and cn.mops has to be adjusted to allow higher copy numbers.

```
> resRef <- referencecn.mops(cases=X[,1],controls=rowMeans(X),
+                classes=c("CN0", "CN1", "CN2", "CN3", "CN4", "CN5", "CN6",
+                                "CN7","CN8","CN16","CN32","CN64","CN128"),
+                I = c(0.025, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 8, 16, 32, 64),
+                segAlgorithm="DNAcopy")

Analyzing: Sample.1

> resRef <- calcIntegerCopyNumbers(resRef)
> (cnvs(resRef))

GRanges object with 4 ranges and 4 metadata columns:
      seqnames       ranges strand | sampleName       median        mean
         <Rle>    <IRanges>  <Rle> | <character>  <numeric>   <numeric>
  [1]    undef    2029-2032      * |      Case_1   0.584963      0.5850
  [2]    undef    2133-2136      * |      Case_1  -5.321928     -5.3219
  [3]    undef    4051-4054      * |      Case_1  -2.660964     -2.6610
  [4]    undef    4528-4533      * |      Case_1  -1.000000     -1.0000
               CN
      <character>
  [1]         CN3
  [2]         CN0
  [3]         CN1
  [4]         CN1
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

8   Heterosomes and CNVs of tumor samples

With the default settings the normalization procedure assumes that the ploidy of each sample is the
same. However, it is possible to account for different karyotypes. When analyzing CNVs on the
X or Y chromosome one possibility is to treat males and females separately. The second option
is to provide the normalization function with the information about the gender, that is different

ploidy states of the X and Y chromosome. This can be handled by the `ploidy` parameter of the normalization function. In the following we show the normalization for the X chromosome, if the first 10 individuals are males (`ploidy` set to 1) and the next 30 individuals are females (`ploidy` set to 2):

```
> XchrX <- normalizeChromosomes(X[1:500, ],ploidy=c(rep(1,10),rep(2,30)))
> cnvr(calcIntegerCopyNumbers(cn.mops(XchrX,norm=FALSE)))
```

```
GRanges object with 1 range and 40 metadata columns:
        seqnames     ranges strand |         S_1         S_2         S_3
           <Rle>  <IRanges>  <Rle> | <character> <character> <character>
  [1]      undef      1-500      * |         CN1         CN1         CN1
                 S_4         S_5         S_6         S_7         S_8
         <character> <character> <character> <character> <character>
  [1]          CN1         CN1         CN1         CN1         CN1
                 S_9        S_10        S_11        S_12        S_13
         <character> <character> <character> <character> <character>
  [1]          CN1         CN1         CN2         CN1         CN2
                S_14        S_15        S_16        S_17        S_18
         <character> <character> <character> <character> <character>
  [1]          CN2         CN1         CN2         CN1         CN2
                S_19        S_20        S_21        S_22        S_23
         <character> <character> <character> <character> <character>
  [1]          CN2         CN2         CN2         CN2         CN2
                S_24        S_25        S_26        S_27        S_28
         <character> <character> <character> <character> <character>
  [1]          CN2         CN2         CN2         CN1         CN1
                S_29        S_30        S_31        S_32        S_33
         <character> <character> <character> <character> <character>
  [1]          CN2         CN2         CN2         CN2         CN2
                S_34        S_35        S_36        S_37        S_38
         <character> <character> <character> <character> <character>
  [1]          CN2         CN2         CN2         CN2         CN0
                S_39        S_40
         <character> <character>
  [1]          CN2         CN1
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Karyotype information can also improve results of CNV detection in tumor samples. The best results can be reached, if for each chromosome the number of appearances in the cell is known. In this case normalization should be applied to each chromosome separately.

9   `cn.mops` **for haploid genomes**

For haploid genomes the prior assumption is that all samples have copy number 1. The function
`haplocn.mops` implements the cn.MOPS algorithm adjusted to haploid genomes.

```
> resHaplo <- haplocn.mops(X)
> resHaplo <- calcIntegerCopyNumbers(resHaplo)
```

10   **Adjusting sensitivity, specificity and resolution for specific applications**

The default parameters of both the local models of `cn.mops` and the segmentation algorithm were
optimized on a wide ranged of different data sets. However, you might want to adjust sensitivity
and specificity or resolution to your specific needs.

`upperThreshold`   The calling threshold for copy number gains, a positive value. Lowering the threshold will
increase the detections, raising will decrease the detections.

`lowerThreshold`   The calling threshold for copy number losses, a negative value. Raising the threshold will
increase the detections, lowering will decrease the detections.

`priorImpact`   This parameter should be optimized for each data set, since it is influenced by number of
samples as well as noise level. The higher the value, the more samples will have copy
number 2, and consequently less CNVs will be detected.

`minWidth`   The minimum length of CNVs measured in number of segments. The more adjacent segments
with a high or low copy number call are joined, the higher the confidence in the
detections. A lower value will lead to more shorter segments, and a higher value will yield
to less, but longer segments.

The length of the initial segments is also crucial. They should be chosen such that on average
50 to 100 reads lie in one segment. The `WL` parameter of `getReadCountsFromBAM` determines this
resolution.

11   **Overview of study designs and `cn.mops` functions**

In Table 1 we give an overview of the functions implemented in this package and present settings
for which they are appropriate. All these functions work for multiple, at least two, samples.CNV
detection for single samples usually yields many false detections, because of characteristics of
genomic segments that lead to a higher or lower read count (and coverage). These biases can
usually not be corrected for (except for the GC content and the mappability bias). Being aware
of all these problems we have implemented the function `singlecn.mops` for cases in which only
one sample is available.

Seq. Type	Ploidy	Study	Samples	Function
WGS	2n	cohort/non-tumor/GWAS	≥ 5	`cn.mops`
WGS	2n	tumor vs. normal	≥ 2	`referencecn.mops`
ES	2n	cohort/non-tumor/GWAS	≥ 5	`exomecn.mops`
ES	2n	tumor vs. normal	≥ 2	`referencecn.mops`
WGS	1n	cohort/non-tumor/GWAS	≥ 5	`haplocn.mops`
WGS	1n	tumor vs. normal	≥ 2	*not implemented*
ES	1n	cohort/non-tumor/GWAS	≥ 5	`haplocn.mops`
ES	1n	tumor vs. normal	≥ 2	*not implemented*

Table 1: **Seq. Type** reports the sequencing technology that was used: whole genome sequencing (WGS) or targeted/exome sequencing (ES). **Ploidy** gives the usual ploidy of the samples. In case of a tumor vs. control study the control sample is meant. **Study**: The type of study to be analyzed for CNVs: either a cohort study, such the HapMap or the 1000Genomes Project, or studies including a number of non-tumor samples or studies with both healthy and diseased individuals, i.e. genome wide association studies (GWAS). **Samples** reports the minimum number of samples needed for the analysis. **Function** gives the function of the `cn.mops` package that is appropriate for this setting.

12 Exporting cn.MOPS results in tabular format

Users can extract the segmentation, the CNVs and the CNV regions with the following:

```
> library(cn.mops); data(cn.mops)
> result <- calcIntegerCopyNumbers(cn.mops(XRanges))
> segm <- as.data.frame(segmentation(result))
> CNVs <- as.data.frame(cnvs(result))
> CNVRegions <- as.data.frame(cnvr(result))
```

The results can be exported with `write.csv` for Excel, LibreOffice Calc, etc. These files will include the genomic position, copy number and other information.

```
> write.csv(segm,file="segmentation.csv")
> write.csv(CNVs,file="cnvs.csv")
> write.csv(CNVRegions,file="cnvr.csv")
```

13 How to cite this package

If you use this package for research that is published later, you are kindly asked to cite it as follows: (Klambauer *et al.*, 2012).

To obtain BibTEX entries of the reference, you can enter the following into your R session:

```
> toBibtex(citation("cn.mops"))
```

References

Alkan, C., Kidd, J. M., Marques-Bonet, T., Aksay, G., Antonacci, F., Hormozdiari, F., Kitzman, J. O., Baker, C., Malig, M., Mutlu, O., Sahinalp, S. C., Gibbs, R. A., and Eichler, E. E. (2009). Personalized copy number and segmental duplication maps using next-generation sequencing. *Nat. Genet.*, **41**(10), 1061–1067.

Chiang, D. Y., Getz, G., Jaffe, D. B., Zhao, X., Carter, S. L., Russ, C., Nusbaum, C., Meyerson, M., and Lander, E. S. (2008). High-resolution mapping of copy-number alterations with massively parallel sequencing. *Nat. Methods*, **6**, 99–103.

Klambauer, G., Schwarzbauer, K., Mitterecker, A., Mayr, A., Clevert, D.-A., Bodenhofer, U., and Hochreiter, S. (2012). cn.MOPS: Mixture of Poissons for Discovering Copy Number Variations in Next Generation Sequencing Data with a Low False Discovery Rate. *Nucleic Acids Research*, **40**(9), e69.

Sathirapongsasuti, F. J., Lee, H., Horst, B. A., Brunner, G., Cochran, A. J., Binder, S., Quackenbush, J., and Nelson, S. F. (2011). Exome Sequencing-Based Copy-Number Variation and Loss of Heterozygosity Detection: ExomeCNV. *Bioinformatics*, **27**(19), 2648–2654.