

# Package ‘myFun’

February 18, 2026

**Type** Package

**Title** myFun is a Collection of My Favourite R Functions, Packaged for Simplicity

**Version** 1.1.1

**Date** 2026-02-18

**Author** Tom Lesluyes [aut, cre]

**Maintainer** Tom Lesluyes <lesluyes.tom@iuct-oncopole.fr>

**Description** My utility functions for R.

**URL** <https://github.com/tlesluyes/myFun>

**BugReports** <https://github.com/tlesluyes/myFun/issues>

**License** GPL-3 + file LICENSE

**Encoding** UTF-8

**LazyData** true

**Depends** R (>= 3.4.0)

**Imports** doParallel,  
foreach,  
GenomeInfoDb,  
GenomicRanges,  
IRanges,  
networkD3,  
rvest,  
S4Vectors,  
stats,  
utils

**Suggests** XVector

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

## R topics documented:

adjustPositions . . . . .	2
BED_metrics . . . . .	3
checkGR . . . . .	4
checkGRlist . . . . .	4
cleanGRlistMetadata . . . . .	5

computeBAF . . . . .	6
computeFit . . . . .	7
computeISA . . . . .	8
computeISA_batch . . . . .	9
computeLogR . . . . .	10
computeMD . . . . .	11
computeMD_batch . . . . .	12
excludeGRanges . . . . .	13
generate_cytoband_and_CHRsize . . . . .	14
get_all_paths . . . . .	14
get_arms . . . . .	15
get_centromeres . . . . .	16
get_Seqinfo . . . . .	17
get_shortest_path . . . . .	17
harmonizeGRanges . . . . .	18
load_CHRsize . . . . .	19
load_cytoband . . . . .	20
occurrenceGRanges . . . . .	20
reestimate_ploidy . . . . .	21
reestimate_purity . . . . .	22
RpackageDependencies . . . . .	23
Rpackages . . . . .	24
splitDF . . . . .	24
summarise_segmetation . . . . .	25

<b>Index</b>	<b>27</b>
--------------	-----------

---

adjustPositions	<i>adjustPositions</i>
-----------------	------------------------

---

## Description

Adjust genomic positions

## Usage

```
adjustPositions(
  DF,
  CHRsize,
  chr_column = "chr",
  start_column = "start",
  end_column = "end",
  suffix = "_adj"
)
```

## Arguments

DF	a data.frame
CHRsize	a data.frame from the load_CHRsize function
chr_column	a column name with chromosome information (default: "chr")
start_column	a column name with start position (default: "start")

end_column	a column name with end position (default: "end"). If end does not exist (e.g. SNP positions), it should be set to the same column as start so it will be ignored
suffix	a suffix for the adjusted positions (default: "_adj")

### Details

This function adjusts genomic positions according to the chromosome sizes. The first nucleotide of chromosome 2 corresponds to the size of the chromosome 1 + 1bp and so on.

### Value

A data.frame with adjusted genomic positions

### Author(s)

tlesluyes

### Examples

```
DF <- data.frame(chr=c(1:3), start=rep(1e6, 3), end=rep(125e6, 3))
load_CHRsize("hg19")
adjustPositions(DF, CHRsize)
```

---

BED_metrics	<i>BED_metrics</i>
-------------	--------------------

---

### Description

Give BED metrics

### Usage

```
BED_metrics(BED, verbose = TRUE)
```

### Arguments

BED	a data.frame, a GRanges object or a path to a BED file
verbose	a boolean, whether to print metrics (default: TRUE)

### Details

This function provides several metrics of interest from a BED file/object.

### Value

A named list of metrics (number of chromosomes, number of regions and total size of the regions) before and after removing overlaps (`GenomicRanges::reduce()`). Strand information is not considered

### Author(s)

tlesluyes

**Examples**

```
# BED format is 0-based for starts!
BED <- data.frame(chr=c(1,1:3)), start=c(0, 1e3, 0, 1e3), end=c(2e3, 5e3, 2e3, 4e3))
BED_metrics(BED)
```

---

checkGR

*checkGR*


---

**Description**

Check that the given object is a valid GRanges object

**Usage**

```
checkGR(myGR, checkOverlaps = TRUE)
```

**Arguments**

myGR                    a GRanges object

checkOverlaps    whether to check for overlapping ranges in the GRanges object (default: TRUE)

**Details**

This function checks that the given object is a valid GRanges object.

**Value**

TRUE if the input is a valid GRanges object

**Author(s)**

tlesluyes

**Examples**

```
GR1 <- GenomicRanges::GRanges(seqnames="1",
                               ranges=IRanges::IRanges(start=c(1, 1001), end=c(1000, 2000)))
checkGR(GR1)
GR2 <- GenomicRanges::GRanges(seqnames="1",
                               ranges=IRanges::IRanges(start=c(1, 500), end=c(1000, 1000)))
checkGR(GR2, checkOverlaps=FALSE)
```

---

checkGRlist	<i>checkGRlist</i>
-------------	--------------------

---

**Description**

Check that the given object is a list of valid GRanges objects

**Usage**

```
checkGRlist(myGRlist, checkOverlaps = TRUE)
```

**Arguments**

myGRlist	a list of GRanges objects
checkOverlaps	whether to check for overlapping ranges in the GRanges objects (default: TRUE)

**Details**

This function checks that the given object is a list of valid GRanges objects.

**Value**

TRUE if the input is a list of valid GRanges objects

**Author(s)**

tlesluyes

**Examples**

```
GR1 <- GenomicRanges::GRanges(seqnames="1", ranges=IRanges::IRanges(start=1, end=1000))
GR2 <- GenomicRanges::GRanges(seqnames="1", ranges=IRanges::IRanges(start=10, end=2000))
checkGRlist(list(GR1, GR2))
```

---

cleanGRlistMetadata	<i>cleanGRlistMetadata</i>
---------------------	----------------------------

---

**Description**

Remove regions with missing metadata in at least one sample from a list of GRanges objects

**Usage**

```
cleanGRlistMetadata(myGRlist, metadataCols, checkOverlaps = TRUE)
```

**Arguments**

myGRlist	a list of harmonized GRanges objects (all granges must be identical across samples)
metadataCols	a character vector of metadata column names to check for missing values
checkOverlaps	whether to check for overlapping ranges in the GRanges objects (default: TRUE)

**Details**

This function removes regions that have missing metadata in at least one sample from a list of GRanges objects.

**Value**

A list of cleaned GRanges objects with the same regions but only those with complete metadata across all samples

**Author(s)**

tlesluyes

**Examples**

```
GR1 <- GenomicRanges::GRanges(seqnames=rep("1", 3),
                              ranges=IRanges::IRanges(start=c(1, 1001, 10001),
                                                         end=c(1000, 10000, 20000)),
                              CNstatus=c(NA, "2+1", "1+1"))
GR2 <- GenomicRanges::GRanges(seqnames=rep("1", 2),
                              ranges=IRanges::IRanges(start=c(500, 10001),
                                                         end=c(10000, 25000)),
                              CNstatus=c("2+1", "1+1"))
myGRList <- harmonizeGRanges(list(GR1=GR1, GR2=GR2), keepHoles=TRUE)
cleanGRListMetadata(myGRList, metadataCols="CNstatus")
```

---

computeBAF

*computeBAF*


---

**Description**

Compute the theoretical BAF values for a given segment

**Usage**

```
computeBAF(nMajor, nMinor, purity, digits = 4)
```

**Arguments**

nMajor	the number of copies of the major allele
nMinor	the number of copies of the minor allele
purity	the purity estimate of the tumour
digits	a numeric, the number of digits to round to (default: 4)

**Details**

This function computes the theoretical BAF values for a given segment (from nMajor, nMinor and purity values).

**Value**

A vector of two numbers representing the BAF values

**Author(s)**

tlesluyes

**See Also**<https://doi.org/10.1038/s41592-020-01013-2>**Examples**

```
# A 2+1 state in a tumour with 90% purity
computeBAF(2, 1, 0.9)
# A 1+0 state in a tumour with 60% purity
computeBAF(1, 0, 0.6)
```

---

computeFit*computeFit*

---

**Description**

Compute the purity/ploidy fit for a given segment

**Usage**

```
computeFit(logR, BAF, nMajor, nMinor, gamma, digits = 4)
```

**Arguments**

logR	the logR value of the segment
BAF	the BAF value of the segment (upper band only so the value should be in the 0.5-1 space)
nMajor	the number of copies of the major allele
nMinor	the number of copies of the minor allele
gamma	the gamma parameter is platform-dependent and represents the expected logR decrease in a diploid sample where one copy is lost (should be 1 for HTS data and 0.55 for SNP arrays)
digits	a numeric, the number of digits to round to (default: 4)

**Details**

This function computes the purity/ploidy fit (rho, psi and psit) for a given segment (from logR, BAF, proposed nMajor and proposed nMinor).

**Value**

A list with the rho (=purity), psi (=total ploidy) and psit (=tumour ploidy) values

**Author(s)**

tlesluyes

See Also

<https://doi.org/10.1038/s41592-020-01013-2>

Examples

```
# A segment has logR=0.5361 and BAF=0.3448/0.6552
# What is the purity/ploidy fit if I believe that the segment is 2+1?
computeFit(0.5361, 0.6552, 2, 1, 1) # purity=90%; ploidy=2
```

---

computeISA	<i>computeISA</i>
------------	-------------------

---

Description

Compute the inter-sample agreement (ISA)

Usage

```
computeISA(GR1, GR2, CNstatus = "CNstatus")
```

Arguments

- GR1                    a GRanges object corresponding to a single CNA profile
- GR2                    a GRanges object corresponding to a single CNA profile
- CNstatus              a metadata column name for the copy-number status (default: "CNstatus"). Can be total (e.g. "3") or allele-specific (e.g. "2+1")

Details

This function computes the inter-sample agreement (ISA) between two profiles (as GRanges objects). This corresponds to the fraction of the genome (%) with the same CN status.

Value

A percentage representing the ISA

Author(s)

tlesluyes

Examples

```
GR1 <- GenomicRanges::GRanges(seqnames=rep("1", 3),
                               ranges=IRanges::IRanges(start=c(1, 1001, 10001),
                                                         end=c(1000, 10000, 20000)),
                               CNstatus=c("1+1", "2+1", "1+1"))
GR2 <- GenomicRanges::GRanges(seqnames=rep("1", 2),
                               ranges=IRanges::IRanges(start=c(500, 10001),
                                                         end=c(10000, 25000)),
                               CNstatus=c("2+1", "1+1"))

# in this example:
#   Region 500-1000 (size=501) is 1+1 for GR1 and 2+1 for GR2
```



```
#   Region 1001-20000 (size=19000) is identical between GR1 and GR2 (both 2+1 and 1+1)
#   ISA is: 19000/19501 = 97.43%
computeISA(GR1, GR2)
```

---

computeISA\_batch

*computeISA\_batch*


---

## Description

Compute the inter-sample agreement (ISA) for a batch of samples

## Usage

```
computeISA_batch(myGRList, cores = 1, min_seg_size = 0, CNstatus = "CNstatus")
```

## Arguments

myGRList	a list of GRanges objects, each object should correspond to one CNA profile
cores	a numeric, the number of cores to use (default: 1)
min_seg_size	a numeric, the minimum segment size (in bp) to consider (default: 0)
CNstatus	a metadata column name for the copy-number status (default: "CNstatus"). Can be total (e.g. "3") or allele-specific (e.g. "2+1")

## Details

This function computes the inter-sample agreement (ISA) between multiple profiles (as a list of GRanges objects).

## Value

A matrix of ISA values

## Author(s)

tlesluyes

## Examples

```
GR1 <- GenomicRanges::GRanges(seqnames=rep("1", 3),
                              ranges=IRanges::IRanges(start=c(1, 1001, 10001),
                                                         end=c(1000, 10000, 20000)),
                              CNstatus=c("1+1", "2+1", "1+1"))
GR2 <- GenomicRanges::GRanges(seqnames=rep("1", 2),
                              ranges=IRanges::IRanges(start=c(500, 10001),
                                                         end=c(10000, 25000)),
                              CNstatus=c("2+1", "1+1"))
GR3 <- GenomicRanges::GRanges(seqnames="1",
                              ranges=IRanges::IRanges(start=500,
                                                         end=25000),
                              CNstatus="1+1")
myGRList <- list(GR1, GR2, GR3)
names(myGRList) <- c("GR1", "GR2", "GR3")
computeISA_batch(myGRList)
```

---

`computeLogR`*computeLogR*

---

**Description**

Compute the theoretical logR value for a given segment

**Usage**

```
computeLogR(nMajor, nMinor, purity, ploidy, digits = 4)
```

**Arguments**

nMajor	the number of copies of the major allele
nMinor	the number of copies of the minor allele
purity	the purity estimate of the tumour
ploidy	the ploidy estimate of the tumour
digits	a numeric, the number of digits to round to (default: 4)

**Details**

This function computes the theoretical logR value for a given segment (from nMajor, nMinor, purity and ploidy values). Since logR isn't allele-specific, ntot can be used instead of nMajor (and nMinor should set to 0).

**Value**

A number representing the logR value

**Author(s)**

tlesluyes

**See Also**

<https://doi.org/10.1038/s41592-020-01013-2>

**Examples**

```
# A 2+1 state in a diploid tumour with 90% purity
computeLogR(2, 1, 0.9, 2)
# A loss of 1 copy (2+1) in a pseudo-tetraploid tumour with 60% purity
computeLogR(2, 1, 0.6, 3.5)
```

---

computeMD

*computeMD*


---

## Description

Compute the Manhattan distance (MD)

## Usage

```
computeMD(GR1, GR2, nMajor = "nMajor", nMinor = "nMinor", convertMb = FALSE)
```

## Arguments

GR1	a GRanges object corresponding to a single CNA profile
GR2	a GRanges object corresponding to a single CNA profile
nMajor	a metadata column name for the major allele (default: "nMajor")
nMinor	a metadata column name for the minor allele (default: "nMinor")
convertMb	a boolean, the MD will be converted to megabases if set to TRUE (default: FALSE)

## Details

This function computes the Manhattan distance (MD) between two profiles (as GRanges objects).

## Value

A numeric value representing the MD

## Author(s)

tlesluyes

## Examples

```
GR1 <- GenomicRanges::GRanges(seqnames=rep("1", 3),
                               ranges=IRanges::IRanges(start=c(1, 1001, 10001),
                                                         end=c(1000, 10000, 20000)),
                               nMajor=c(1, 2, 1),
                               nMinor=c(1, 1, 1))
GR2 <- GenomicRanges::GRanges(seqnames=rep("1", 2),
                               ranges=IRanges::IRanges(start=c(500, 10001),
                                                         end=c(10000, 25000)),
                               nMajor=c(2, 1),
                               nMinor=c(1, 1))

# in this example:
#   Region 500-1000 (size=501) is 1+1 for GR1 and 2+1 for GR2
#   Region 1001-20000 (size=19000) is identical between GR1 and GR2 (both 2+1 and 1+1)
#   MD is: (abs(2-1)+abs(1-1))*501 = 501
computeMD(GR1, GR2)
```



```

                                nMajor=c(2, 1),
                                nMinor=c(1, 1))
GR3 <- GenomicRanges::GRanges(seqnames="1",
                                ranges=IRanges::IRanges(start=500,
                                                            end=25000),
                                nMajor=1,
                                nMinor=1)
myGRList <- list(GR1, GR2, GR3)
names(myGRList) <- c("GR1", "GR2", "GR3")
computeMD_batch(myGRList)

```

---

excludeGRanges

*excludeGRanges*


---

## Description

Exclude GRanges regions

## Usage

```
excludeGRanges(GR.ref, GR.toremove)
```

## Arguments

GR.ref	a GRanges object to be filtered
GR.toremove	a GRanges object containing regions to exclude

## Details

This function excludes GRanges regions from a reference GRanges object.

## Value

A filtered GRanges object

## Author(s)

tlesluyes

## Examples

```

GR1 <- GenomicRanges::GRanges(seqnames = rep(1, 2),
                                ranges=IRanges::IRanges(start=c(1, 5e5+1), end=c(5e5, 1e6)),
                                score=c(3, 2))
GR2 <- GenomicRanges::GRanges(seqnames = 1,
                                ranges=IRanges::IRanges(start=4e5+1, end=6e5))
excludeGRanges(GR1, GR2)

```

---

```
generate_cytoband_and_CHRsize
    generate_cytoband_and_CHRsize
```

---

**Description**

Generate cytoband and CHRsize information

**Usage**

```
generate_cytoband_and_CHRsize(cytoband_file)
```

**Arguments**

cytoband\_file    a cytoband file

**Details**

This function generates cytoband and CHRsize information from a cytoband file. This can be obtained from the UCSC table browser -> select a genome/assembly -> "Mapping and Sequencing" -> "Chromosome Band" (not the ideogram version!) -> "get output" -> Remove the first "#" character (keep the header!).

**Value**

A list with both the cytoband and CHRsize information

**Author(s)**

tlesluyes

**See Also**

```
load_CHRsize("hg38"); load_cytoband("hg38")
```

---

```
get_all_paths                      get_all_paths
```

---

**Description**

Get all possibles paths between two copy-number states

**Usage**

```
get_all_paths(start, end, WGD, max_path_size = 5, simplify = TRUE)
```

**Arguments**

start	a vector of length 2 (representing a copy-number state; e.g. c(1, 1) represents a 1+1 state), defining where to start
end	a vector of length 2 (representing a copy-number state; e.g. c(1, 1) represents a 1+1 state), defining where to end
WGD	a boolean defining if WGD events are allowed
max_path_size	an integer defining the maximum path size
simplify	a boolean defining if consecutive and opposite alterations (e.g. +1/+0 and then -1/-0) are allowed

**Details**

This function returns all possible paths between two copy-number states. The expected input is allele-specific (with two values), but it can be used for total copy-number by setting c(ntot, 0). Possible events include: +1/+0 (gain of the major allele), -1/-0 (loss of the major allele), +0/+1 (gain of the minor allele), -0/-1 (loss of the minor allele) and WGD.

**Value**

A vector of all possible paths given as characters (separator=";")

**Author(s)**

tlesluyes

**Examples**

```
# Diploid baseline (1+1) turns into 2+1
print(get_all_paths(start=c(1, 1), end=c(2, 1), WGD=TRUE))
# Chromosome X in males (1+0) is gained (5 copies)
print(get_all_paths(start=c(1, 0), end=c(5, 0), WGD=TRUE))
```

---

get\_arms

get\_arms

---

**Description**

Get chromosome arms information

**Usage**

```
get_arms(assembly, withCentromeres = TRUE)
```

**Arguments**

assembly	an assembly (e.g. hg38) or a data.frame with expected cytoband information (chr, start, end, name, gieStain)
withCentromeres	whether to include centromeric regions (default: TRUE)

**Details**

This function gets chromosome arms information for a given assembly.

**Value**

A data.frame with genomic regions, can be converted to GRanges on the fly

**Author(s)**

tlesluyes

**Examples**

```
get_arms("hg38")
```

---

get\_centromeres

*get\_centromeres*

---

**Description**

Get centromeres information

**Usage**

```
get_centromeres(assembly)
```

**Arguments**

assembly	an assembly (e.g. hg38) or a data.frame with expected cytoband information (chr, start, end, gieStain)
----------	--

**Details**

This function gets centromeres information for a given assembly.

**Value**

A data.frame with genomic regions, can be converted to GRanges on the fly

**Author(s)**

tlesluyes

**Examples**

```
get_centromeres("hg38")
```



---

get_Seqinfo	<i>get_Seqinfo</i>
-------------	--------------------

---

**Description**

Get Seqinfo object

**Usage**

```
get_Seqinfo(assembly, genome = NA)
```

**Arguments**

assembly	an assembly (e.g. hg38) or a data.frame with expected cytoband information (chr, size)
genome	genome name (e.g. hg38), required if assembly is a data.frame (default: NA)

**Details**

This function gets Seqinfo object for a given assembly.

**Value**

A Seqinfo object

**Author(s)**

tlesluyes

**Examples**

```
get_Seqinfo("hg38")
```

---

get_shortest_path	<i>get_shortest_path</i>
-------------------	--------------------------

---

**Description**

Get the shortest path among several

**Usage**

```
get_shortest_path(paths, wanted_WGD = NA, count_WGD = FALSE)
```

**Arguments**

paths	all possible paths to consider
wanted_WGD	a numeric value defining the number of WGD events wanted (can be NA to allow for any possibility, including no event at all; default: NA)
count_WGD	a boolean defining if the number of WGD events should be counted (default: FALSE)

**Details**

This function returns the shortest possible path. It should be used after running the `get_all_paths` function or can be used as long as the input format is correct.

**Value**

A numeric value representing the minimal number of events, its name represents the full path

**Author(s)**

tlesluyes

**Examples**

```
# Diploid baseline (1+1) turns into 2+1
print(get_shortest_path(get_all_paths(start=c(1, 1), end=c(2, 1), WGD=TRUE)))
# Chromosome X in males (1+0) is gained (5 copies)
print(get_shortest_path(get_all_paths(start=c(1, 0), end=c(5, 0), WGD=TRUE)))
```

---

harmonizeGRanges	<i>harmonizeGRanges</i>
------------------	-------------------------

---

**Description**

Harmonize GRanges objects

**Usage**

```
harmonizeGRanges(myGRList, cores = 1, keepHoles = FALSE)
```

**Arguments**

myGRList	a list of GRanges objects, each object should correspond to one CNA profile
cores	a numeric, the number of cores to use (default: 1)
keepHoles	a logical, whether to keep holes (regions not covered by all samples) in the output (default: FALSE)

**Details**

This function harmonizes GRanges objects by keeping only regions covered by all samples.

**Value**

A list of harmonized GRanges objects

**Author(s)**

tlesluyes

**Examples**

```
GR1 <- GenomicRanges::GRanges(seqnames="1",
                               ranges=IRanges::IRanges(start=1, end=1000),
                               nMajor=1, nMinor=1)
GR2 <- GenomicRanges::GRanges(seqnames="1",
                               ranges=IRanges::IRanges(start=10, end=2000),
                               nMajor=2, nMinor=1)
harmonizeGRanges(list(GR1, GR2))
```

---

load_CHRsize	<i>load_CHRsize</i>
--------------	---------------------

---

**Description**

Load CHRsize information

**Usage**

```
load_CHRsize(assembly)
```

**Arguments**

assembly            an assembly (hg19, hg38 or CHM13)

**Details**

This function loads CHRsize information for a given assembly. It is then available as a data.frame called CHRsize in the environment.

**Value**

A data.frame with the CHRsize information

**Author(s)**

tlesluyes

**Examples**

```
load_CHRsize("hg38"); head(CHRsize)
```

---

load_cytoband	<i>load_cytoband</i>
---------------	----------------------

---

**Description**

Load cytoband information

**Usage**

```
load_cytoband(assembly)
```

**Arguments**

assembly	an assembly (hg19, hg38 or CHM13)
----------	-----------------------------------

**Details**

This function loads cytoband information for a given assembly. It is then available as a data.frame called cytoband in the environment.

**Value**

A data.frame with the cytoband information

**Author(s)**

tlesluyes

**Examples**

```
load_cytoband("hg38"); head(cytoband)
```

---

occurrenceGRanges	<i>occurrenceGRanges</i>
-------------------	--------------------------

---

**Description**

Get the occurrence of events

**Usage**

```
occurrenceGRanges(myGRList, myMetadata, computeP = TRUE)
```

**Arguments**

myGRList	a list of GRanges objects, each object should correspond to one CNA profile
myMetadata	a vector of metadata to consider
computeP	whether to compute the percentage of samples with specific metadata for each region (default: TRUE)

**Details**

This function gets the occurrence of events in a list of GRanges objects. All objects must have the same metadata columns and metadata must be TRUE/FALSE.

**Value**

A GRanges object with nSamples as the total number of samples and metadata columns with the occurrence of events

**Author(s)**

tlesluyes

**Examples**

```
GR1 <- GenomicRanges::GRanges(seqnames="1",
                              ranges=IRanges::IRanges(start=1, end=1000),
                              Gain=TRUE, Loss=FALSE)
GR2 <- GenomicRanges::GRanges(seqnames="1",
                              ranges=IRanges::IRanges(start=10, end=2000),
                              Gain=FALSE, Loss=TRUE)
occurrenceGRanges(list(GR1, GR2), c("Gain", "Loss"))
```

---

reestimate_ploidy	<i>reestimate_ploidy</i>
-------------------	--------------------------

---

**Description**

Compute the re-estimated ploidy for a given sample

**Usage**

```
reestimate_ploidy(rho.old, psit.old, rho.new, WGD, digits = 4)
```

**Arguments**

rho.old	old purity estimate
psit.old	old ploidy estimate
rho.new	new purity estimate
WGD	number of WGD events (0 if there is no WGD)
digits	a numeric, the number of digits to round to (default: 4)

**Details**

This function computes the re-estimated ploidy for a given sample (from its old purity/ploidy fit and the re-estimated purity).

**Value**

A number representing the re-estimated ploidy

**Author(s)**

tlesluyes

**Examples**

```
# A pseudo-diploid sample has purity=74% and ploidy=2.4
# What is the re-estimated ploidy if I believe that the sample has purity=61%?
reestimate_ploidy(0.74, 2.4, 0.61, 0)
```

---

reestimate_purity	<i>reestimate_purity</i>
-------------------	--------------------------

---

**Description**

Compute the re-estimated purity for a given sample

**Usage**

```
reestimate_purity(rho.old, psit.old, switch, digits = 4)
```

**Arguments**

rho.old	old purity estimate
psit.old	old ploidy estimate
switch	a character ("double" or "halve") indicating whether the ploidy should be doubled or halved
digits	a numeric, the number of digits to round to (default: 4)

**Details**

This function computes the re-estimated purity for a given sample in the context of a jump in ploidy (so the matched ploidy needs to be doubled or halved).

**Value**

A number representing the re-estimated purity

**Author(s)**

tlesluyes

**Examples**

```
# A sample has purity=74% and ploidy=2.4 but the CNA profile needs to be doubled
# What is the re-estimated purity?
reestimate_purity(0.74, 2.4, "double")
```

---

RpackageDependencies    *RpackageDependencies*

---

### Description

Show the package dependencies

### Usage

```
RpackageDependencies(  
  customFolder = NULL,  
  customDependencyTypes = NULL,  
  customColours = NULL,  
  simplifyNetwork = TRUE,  
  saveFile = NULL  
)
```

### Arguments

customFolder	a vector of folder names (default: NULL; .libPaths() is used)
customDependencyTypes	a vector of dependency types, possible values are: "Depends", "Imports", "LinkingTo", "Suggests" and "Enhances" (default: c("Depends", "Imports", "LinkingTo"))
customColours	a named vector of colours. Names must correspond to the dependency types and values must be valid colours (default: NULL; an internal colour scheme is used)
simplifyNetwork	a boolean defining if the network should be simplified, i.e. the R base packages are removed (default: TRUE)
saveFile	a string defining the name of the HTML file where the network should be saved (default: NULL; no file is saved)

### Details

Given a folder of R packages, this function reads the DESCRIPTION files of the installed packages and shows their dependencies.

### Value

A list with nodes (a data.frame of R packages), links (a data.frame of package dependencies) and plot (a network plot using networkD3)

### Author(s)

tlesluyes

### Examples

```
myDep <- RpackageDependencies()  
print(head(myDep$nodes))  
print(head(myDep$links))
```

---

Rpackages	<i>Rpackages</i>
-----------	------------------

---

**Description**

List installed packages and determine their source

**Usage**

```
Rpackages(
  CRAN_URL = "http://cran.us.r-project.org",
  Bioconductor_URL = "https://www.bioconductor.org/packages/release/bioc/"
)
```

**Arguments**

CRAN\_URL            the CRAN URL (default: "http://cran.us.r-project.org")  
 Bioconductor\_URL            the Bioconductor URL (default: "https://www.bioconductor.org/packages/release/bioc/")

**Details**

This function lists installed packages and determine whether they are base packages or come from CRAN/Bioconductor or if they are external (GitHub, SourceForge, etc.). This function requires an internet connection.

**Value**

A data.frame with the installed packages and an additional column: Source (possible values: Base, CRAN, Bioconductor, External)

**Author(s)**

tlesluyes

**Examples**

```
head(Rpackages())
```

---

splitDF	<i>splitDF</i>
---------	----------------

---

**Description**

Split a data.frame

**Usage**

```
splitDF(DF, chunks, shuffle = FALSE, seed = 1234)
```



**Arguments**

DF	a data.frame to split
chunks	a number of chunks to obtain
shuffle	a boolean, whether to shuffle the data.frame before splitting (default: FALSE)
seed	a number, the seed for the random number generator (default: 1234)

**Details**

This function splits a data.frame into a list of data.frames.

**Value**

A list of data.frames

**Author(s)**

tlesluyes

**Examples**

```
DF <- data.frame(a=1:26, b=letters)
splitDF(DF, 3)
```

---

summarise\_segmetation    *summarise\_segmetation*

---

**Description**

Summarise segmentation data

**Usage**

```
summarise_segmetation(DF, col_chr, col_start, col_end, col_values)
```

**Arguments**

DF	a data.frame with segmentation data
col_chr	a string, the name of the column containing the chromosome
col_start	a string, the name of the column containing the start position
col_end	a string, the name of the column containing the end position (can be the same as col_start for SNP-based segmentation where start=end)
col_values	a vector of strings, the names of the columns containing the values of interest (logR, BAF, etc.)

**Details**

This function summarises segmentation data, typically logR and/or BAF values for individual SNPs or loci.

**Value**

A named list with segments being a data.frame with the summarised information and IDs being a list of SNPs/loci associated with the different segments

**Author(s)**

tlesluyes

**Examples**

```
DF <- data.frame(chr=c(rep("chr1", 10),rep("chr2", 6)),
                 pos=c(1:10*1e3, 1:6*1e3),
                 logR=c(rep(0, 4), rep(0.54, 3), rep(0, 3), rep(-0.86, 3), rep(0, 3)),
                 BAF=c(rep(0.5, 4), rep(0.34, 3), rep(0.5, 3), rep(0.09, 3), rep(0.5, 3)),
                 row.names=paste0("SNP_", 1:16))

DF
summarise_segmetation(DF, "chr", "pos", "pos", c("logR", "BAF"))
```

# Index

`adjustPositions`, [2](#)

`BED_metrics`, [3](#)

`checkGR`, [4](#)  
`checkGRlist`, [4](#)  
`cleanGRlistMetadata`, [5](#)  
`computeBAF`, [6](#)  
`computeFit`, [7](#)  
`computeISA`, [8](#)  
`computeISA_batch`, [9](#)  
`computeLogR`, [10](#)  
`computeMD`, [11](#)  
`computeMD_batch`, [12](#)

`excludeGRanges`, [13](#)

`generate_cytoband_and_CHRsize`, [14](#)  
`get_all_paths`, [14](#)  
`get_arms`, [15](#)  
`get_centromeres`, [16](#)  
`get_Seqinfo`, [17](#)  
`get_shortest_path`, [17](#)

`harmonizeGRanges`, [18](#)

`load_CHRsize`, [19](#)  
`load_cytoband`, [20](#)

`occurrenceGRanges`, [20](#)

`reestimate_ploidy`, [21](#)  
`reestimate_purity`, [22](#)  
`RpackageDependencies`, [23](#)  
`Rpackages`, [24](#)

`splitDF`, [24](#)  
`summarise_segmetation`, [25](#)