Introduction to *MutationalPatterns*

Bastiaan Van der Roest [1], ***Francis Blokzijl*** [1], ***Roel Janssen*** [1], ***Ruben van Boxtel*** [1], ***and Edwin Cuppen*** [1]

[1]University Medical Center Utrecht, Utrecht, The Netherlands

December 20, 2019

Contents

1 Introduction

Mutational processes leave characteristic footprints in genomic DNA. This package provides a comprehensive set of flexible functions that allows researchers to easily evaluate and visualize a multitude of mutational patterns in base substitution catalogues of e.g. tumour samples or DNA-repair deficient cells. The package covers a wide range of patterns including: mutational signatures, transcriptional and replicative strand bias, genomic distribution and association with genomic features, which are collectively meaningful for studying the activity of mutational processes. The package provides functionalities for both extracting mutational signatures *de novo* and determining the contribution of previously identified mutational signatures on a single sample level. MutationalPatterns integrates with common R genomic analysis workflows and allows easy association with (publicly available) annotation data.

Background on the biological relevance of the different mutational patterns, a practical illustration of the package functionalities, comparison with similar tools and software packages and an elaborate discussion, are described in the MutationalPatterns article, which is published in Genome Medicine in 2018: https://doi.org/10.1186/s13073-018-0539-0

2 Data

To perform the mutational pattern analyses, you need to load one or multiple VCF files with substitutions and/or indel calls and the corresponding reference genome.

2.1 List reference genome

List available genomes using *BSgenome*:

```
> library(BSgenome)
> head(available.genomes())
```

```
[1] "BSgenome.Alyrata.JGI.v1"                "BSgenome.Amellifera.BeeBase.assembly4"
[3] "BSgenome.Amellifera.UCSC.apiMel2"       "BSgenome.Amellifera.UCSC.apiMel2.masked"
[5] "BSgenome.Athaliana.TAIR.04232008"       "BSgenome.Athaliana.TAIR.TAIR9"
```

Download and load your reference genome of interest:

```
> ref_genome <- "BSgenome.Hsapiens.UCSC.hg19"
> library(ref_genome, character.only = TRUE)
```

2.2 Load example data

We provided an example data set with this package, which consists of a subset of somatic mutation catalogues of 9 normal human adult stem cells from 3 different tissues (Blokzijl et al., 2016). When own data is loaded, please pay attention that the files are in VCF format 4.2 or higher, which makes sure that all variants are loaded correctly.

Load the MutationalPatterns package:

```
> #library(MutationalPatterns)
> devtools::load_all("../R", export_all = FALSE)
```

Locate the VCF files of the example data:

```
> vcf_files <- list.files(system.file("extdata", package="MutationalPatterns"),
+                         pattern = ".vcf", full.names = TRUE)
> vcf_files <- vcf_files[4:12]
```

Define corresponding sample names for the VCF files:

```
> sample_names <- c(
+   "colon1", "colon2", "colon3",
+   "intestine1", "intestine2", "intestine3",
+   "liver1", "liver2", "liver3")
```

Load the VCF files into a `GRangesList`:

```
> vcfs <- read_vcfs_as_granges(vcf_files, sample_names, ref_genome,
+                              group = "auto+sex", check_alleles = TRUE)
```

```
           Number of SNV Number of DBS Number of indel
colon1               200             0               0
colon2               598             1               0
colon3               450             0               0
intestine1           150             0               0
intestine2           800             0               0
intestine3           500             0               0
liver1               300             0               0
liver2               896             2               0
liver3               198             1               0
```

```
> summary(vcfs)
```

```
     Length       Class        Mode
          9 GRangesList          S4
```

Define relevant metadata on the samples, such as tissue type:

```
> tissue <- c(rep("colon", 3), rep("intestine", 3), rep("liver", 3))
```

3 Mutation characteristics

3.1 Single base substitution types

We can retrieve base substitutions from the VCF GRanges object as "REF>ALT" using `mutations_from_vcf`:

```
> muts = mutations_from_vcf(vcfs[[1]])
> head(muts, 12)
```

```
 [1] "T>A" "T>C" "G>A" "A>C" "G>A" "A>G" "C>T" "A>G" "G>T" "A>G" "G>A" "G>A"
```

We can retrieve the base substitutions from the VCF GRanges object and convert them to the 6 types of base substitution types that are distinguished by convention: C>A, C>G, C>T, T>A, T>C, T>G. For example, when the reference allele is G and the alternative allele is T (G>T), `mut_type` returns the G:C>T:A mutation as a C>A mutation:

```
> types = mut_type(vcfs[[1]])
> head(types, 12)
```

```
 [1] "T>A" "T>C" "C>T" "T>G" "C>T" "T>C" "C>T" "T>C" "C>A" "T>C" "C>T" "C>T"
```

To retrieve the sequence context (one base upstream and one base downstream) of the single base substitutions in the VCF object from the reference genome, you can use the `mut_context` function:

```
> context = mut_context(vcfs[[1]], ref_genome)
> head(context, 12)
```

```
 chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr2  chr2  chr2
"GTT" "ATT" "CGC" "CAG" "AGC" "AAC" "ACA" "AAG" "TGA" "GAG" "CGT" "CGA"
```

With `type_context`, you can retrieve the types and contexts for all positions in the VCF GRanges object. For the base substitutions that are converted to the conventional base substitution types, the reverse complement of the sequence context is returned.

```
> type_context = type_context(vcfs[[1]], ref_genome)
> lapply(type_context, head, 12)

$types
 [1] "T>A" "T>C" "C>T" "T>G" "C>T" "T>C" "C>T" "T>C" "C>A" "T>C" "C>T" "C>T"

$context
 chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr2  chr2  chr2
"GTT" "ATT" "GCG" "CTG" "GCT" "GTT" "ACA" "CTT" "TCA" "CTC" "ACG" "TCG"
```

With `mut_type_occurrences`, you can count mutation type occurrences for all VCF objects in the `GRangesList`. For C>T mutations, a distinction is made between C>T at CpG sites and other sites, as deamination of methylated cytosine at CpG sites is a common mutational process. For this reason, the reference genome is needed for this functionality.

```
> type_occurrences <- mut_type_occurrences(vcfs, ref_genome)
> type_occurrences

           C>A C>G C>T T>A T>C T>G C>T at CpG C>T other
colon1      28   5 111  13  31  12         59        52
colon2      77  29 345  36  90  21        151       194
colon3      79  19 243  25  61  23        165        78
intestine1  19   8  74  19  26   4         33        41
intestine2 118  49 423  57 126  27        258       165
intestine3  54  27 298  32  67  22        192       106
liver1      43  22  94  30  77  34         18        76
liver2     144  93 274 103 209  73         20       254
liver3      39  28  61  15  32  23          4        57
```

3.2 Mutation spectrum

A mutation spectrum shows the relative contribution of each mutation type in the base substitution catalogs. The `plot_spectrum` function plots the mean relative contribution of each of the 6 base substitution types over all samples. Error bars indicate standard deviation over all samples. The total number of mutations is indicated.

```
> p1 <- plot_spectrum(type_occurrences)
```

Plot the mutation spectrum with distinction between C>T at CpG sites and other sites:

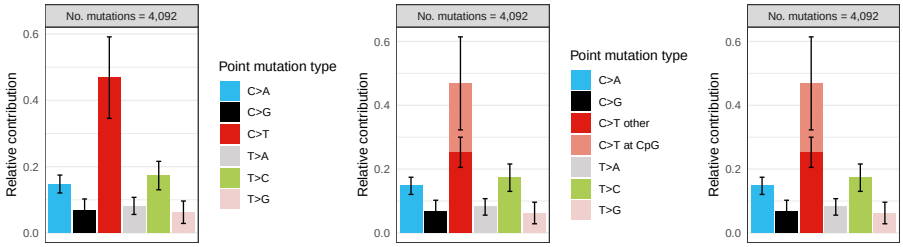
```
> p2 <- plot_spectrum(type_occurrences, CT = TRUE)
```

Plot spectrum without legend:

```
> p3 <- plot_spectrum(type_occurrences, CT = TRUE, legend = FALSE)
```

The gridExtra package will be used throughout this vignette to combine multiple plots:

```
> library("gridExtra")
> grid.arrange(p1, p2, p3, ncol=3, widths=c(3,3,1.75))
```



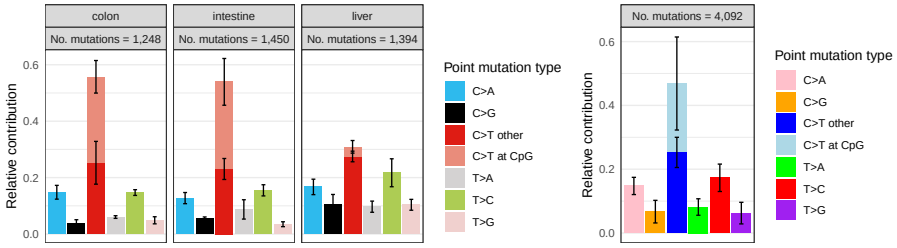
You can facet the per sample group, e.g. plot the spectrum for each tissue separately:

```
> p4 <- plot_spectrum(type_occurrences, by = tissue, CT = TRUE, legend = TRUE)
```

Define your own 7 colors for spectrum plotting:

```
> palette <- c("pink", "orange", "blue", "lightblue", "green", "red", "purple")
> p5 <- plot_spectrum(type_occurrences, CT=TRUE, legend=TRUE, colors=palette)
```

```
> grid.arrange(p4, p5, ncol=2, widths=c(4,2.3))
```



3.3 Double base substitutions and indels

Not only single base substitutions can be retrieved from the VCF GRanges object, also double base substitutions and/or indels can be extracted, if they are present in the loaded VCF files. Double base substitutions have the format "REF:NN > ALT:NN" or they are two SNVs with consecutive positions. Indels must be in at least VCF format 4.2. That means that deletions have a REF with the deletion length and an ALT with length 1, and insertions have a REF of length 1 and an ALT with the insertion length. Moreover, the REF and ALT of indels only contains nucleotide letters (A, C, G and T), no other characters.

These two types of mutations are retrieved the same way as the single base substitutions: "REF>ALT", using `mutations_from_vcf`. Therefore set the argument `type` to a vector of the wanted mutation types. When multiple mutation types are requested, the output will be a list of mutation types.

For the double base substitutions and indels other data is used, which contain more variants of these mutation types. The data is from breast cancer organoids, described in the work of Sachs et al. ((Sachs et al., 2018)).

First store the old vcfs, in order to use them downstream:

```
> vcfs_tissues = vcfs
```

Then load the new data:

```
> vcf_files = list.files(system.file("extdata", package="MutationalPatterns"),
+                        pattern = ".vcf", full.names = TRUE)
> vcf_files = vcf_files[1:3]
> sample_names = paste0("breast",1:3)
> vcfs_breast = read_vcfs_as_granges(vcf_files, sample_names, ref_genome, group = "auto+sex",
+                            check_alleles = TRUE, dbs_format = "one-line")
```

```
        Number of SNV Number of DBS Number of indel
breast1          3627            23            1061
breast2         12083            65            2279
breast3          4722            37            1536
```

```
> muts = mutations_from_vcf(vcfs_breast[[1]], type = c("dbs", "indel"))
> lapply(muts, head, 12)
```

```
$dbs
 [1] "AT>GG" "CA>TG" "TG>CA" "GA>TT" "CA>GC" "AT>GG" "TG>CA" "AT>GC" "GT>TG" "TT>GG" "GC>AT"
[12] "TG>CA"

$indel
 [1] "TTCTC>T"           "TTATA>T"           "TTC>T"             "CCCACCCATCCAT>C"
 [5] "TTG>T"             "C>CT"              "GCTGAAAAC>G"       "T>TACAGTCTGTTGAGC"
 [9] "G>GA"              "ATC>A"             "GTA>G"             "AC>A"
```

To convert the double base substitutions to the 78 strand-agnostic types found in the COSMIC database, run the function `mut_type`. The 1 basepair indels will also be converted to a "C" or "T" indel with this function:

```
> types = mut_type(vcfs_breast[[1]], type = c("dbs", "indel"))
> lapply(types, head, 12)
```

```
$dbs
 [1] "AT>CC" "TG>CA" "TG>CA" "TC>AA" "TG>GC" "AT>CC" "TG>CA" "AT>GC" "AC>CA" "TT>GG" "GC>AT"
[12] "TG>CA"

$indel
 [1] "TTCTC>T"           "TTATA>T"           "TTC>T"             "CCCACCCATCCAT>C"
 [5] "TTG>T"             "C>CT"              "GCTGAAAAC>G"       "T>TACAGTCTGTTGAGC"
 [9] "G>GA"              "ATC>A"             "GTA>G"             "AC>A"
```

The insertions and deletions can be translated to a more clear definition, on which the indels can be grouped. Since there is no single intuitive and naturally constrained set of indel mutation types, it is possible to give an own definition of indels and to set global variables for this definition. For this the function `indel_mutation_type` can be used. To set the indel context following the COSMIC database, the default option, use:

```
> indel_mutation_type("cosmic")
```

Then the indel mutations can be translated with `mut_context`:

```
> context = mut_context(vcfs_breast[[1]], ref_genome, type = "indel")
> head(context, 12)

 [1] "del.rep.len.4.rep.6+"    "del.rep.len.4.rep.3"     "del.rep.len.2.rep.6+"
 [4] "del.rep.len.5+.rep.2"    "del.mh.len.2.bimh.1"     "ins.1bp.homopol.T.len.1"
 [7] "del.rep.len.5+.rep.1"    "ins.rep.len.5+.rep.1"    "ins.1bp.homopol.T.len.0"
[10] "del.rep.len.2.rep.6+"    "del.rep.len.2.rep.6+"    "del.1bp.homopol.C.len.4"
```

As with the single base substitutions, `type_context` can be used to retrieve type and context information of all double base substitutions, insertions and deletions. The function will return the type and context information as a list of mutation types:

```
> type_context = type_context(vcfs_breast[[1]], ref_genome, type = c("dbs","indel"))
> lapply(type_context, function(x) lapply(x, head, 10))

$dbs
$dbs$types
 [1] "AT>CC" "TG>CA" "TG>CA" "TC>AA" "TG>GC" "AT>CC" "TG>CA" "AT>GC" "AC>CA" "TT>GG"


$indel
$indel$types
 [1] "TTCTC>T"           "TTATA>T"           "TTC>T"             "CCCACCCATCCAT>C"
 [5] "TTG>T"             "C>CT"              "GCTGAAAAC>G"       "T>TACAGTCTGTTGAGC"
 [9] "G>GA"              "ATC>A"

$indel$context
 [1] "del.rep.len.4.rep.6+"    "del.rep.len.4.rep.3"     "del.rep.len.2.rep.6+"
 [4] "del.rep.len.5+.rep.2"    "del.mh.len.2.bimh.1"     "ins.1bp.homopol.T.len.1"
 [7] "del.rep.len.5+.rep.1"    "ins.rep.len.5+.rep.1"    "ins.1bp.homopol.T.len.0"
[10] "del.rep.len.2.rep.6+"
```

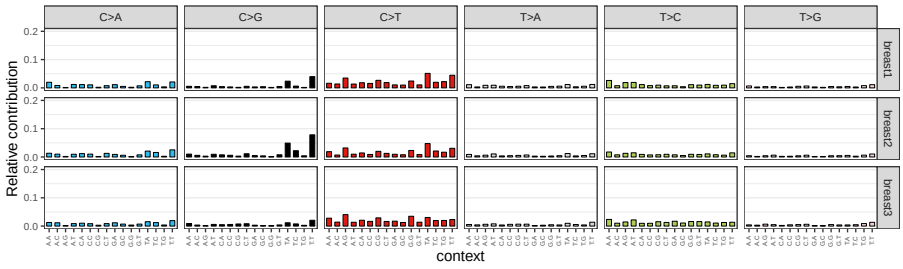
3.4 Mutational profiles

Make a 96 trinucleodide mutation count matrix:

```
> mut_mat <- mut_matrix(vcf_list = vcfs_breast, ref_genome = ref_genome)
> head(mut_mat)

        breast1 breast2 breast3
A[C>A]A      72     159      60
A[C>A]C      31     121      55
A[C>A]G       5      23       9
A[C>A]T      44     117      41
C[C>A]A      42     144      48
C[C>A]C      39     122      40
```

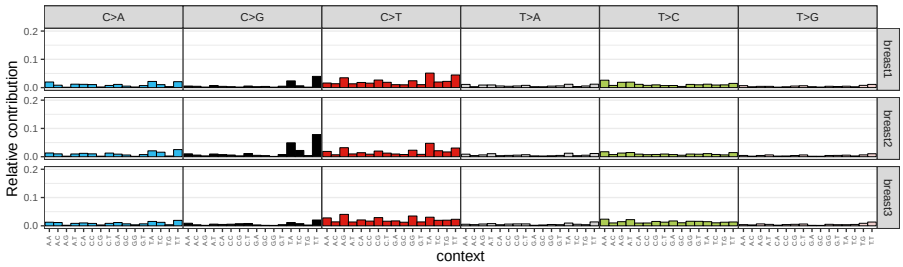
Plot the 96 profile of two samples:

```
> plot_profiles(mut_mat)
```



Plot 96 profile of two samples in a more condensed plotting format:

```
> plot_profiles(mut_mat, condensed = TRUE)
```



To plot the mutation profiles of different mutation types (SBS, DBS and/or indels), first make a list of mutation count matrices:

```
> mut_mat <- mut_matrix(vcf_list = vcfs_breast, ref_genome = ref_genome, type = "all")
> lapply(mut_mat, head)

$snv
        breast1 breast2 breast3
A[C>A]A      72     159      60
A[C>A]C      31     121      55
A[C>A]G       5      23       9
A[C>A]T      44     117      41
C[C>A]A      42     144      48
C[C>A]C      39     122      40

$dbs
      breast1 breast2 breast3
AC>CA       1       2       1
AC>CG       0       0       0
AC>CT       1       1       0
AC>GA       0       0       0
AC>GG       0       0       0
AC>GT       0       1       2

$indel
```

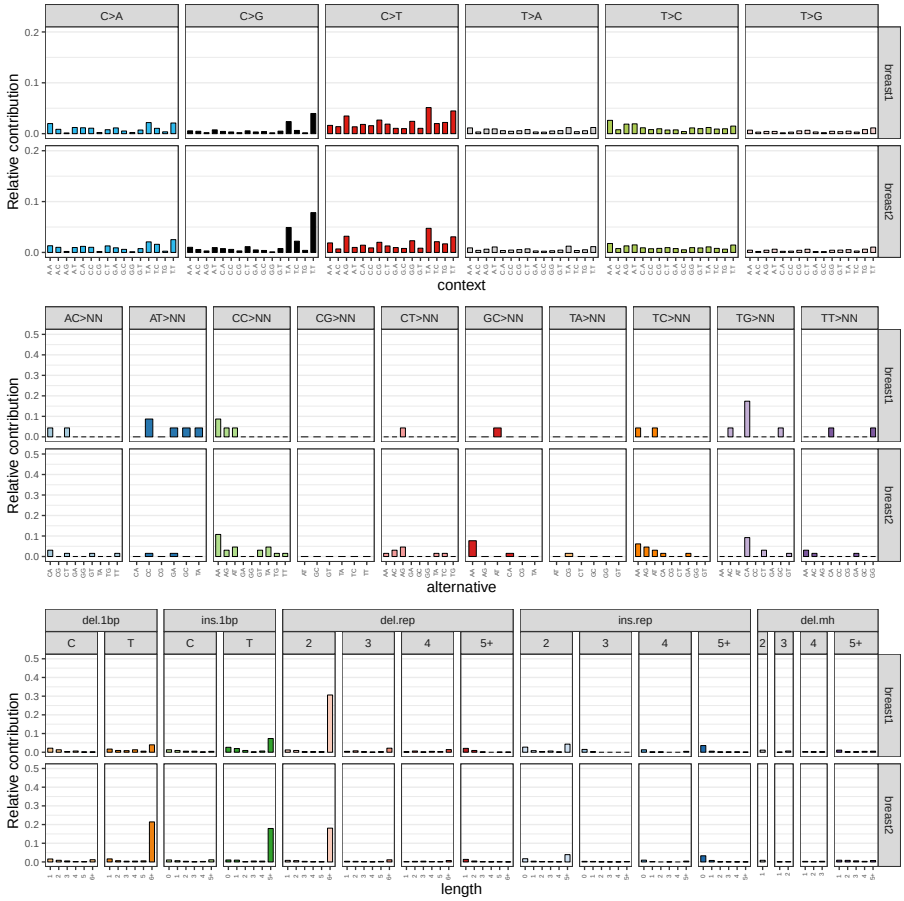
```
                          breast1 breast2 breast3
del.1bp.homopol.C.len.1        23      36      30
del.1bp.homopol.C.len.2        14      20      18
del.1bp.homopol.C.len.3         4      12      13
del.1bp.homopol.C.len.4         7       5       7
del.1bp.homopol.C.len.5         2       4       5
del.1bp.homopol.C.len.6+        2      27       5
```

Make a list of two samples:

```
> mut_mat_sub <- list("snv" = mut_mat$snv[,c(1,2)],
+                     "dbs" = mut_mat$dbs[,c(1,2)],
+                     "indel" = mut_mat$indel[,c(1,2)])
```

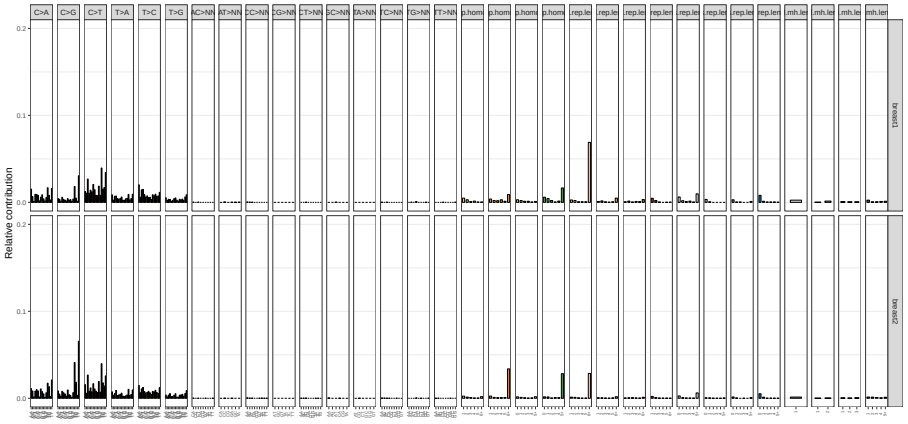
Plot the mutation profiles of the two samples:

```
> plot_profiles(mut_mat_sub)
```



It is also possible to plot mutation profiles with all mutation types together.

```
> plot_profiles(mut_mat_sub, method = "combine")
```



4 Mutational signatures

4.1 *De novo* mutational signature extraction using NMF

Mutational signatures are thought to represent mutational processes, and are characterized by a specific contribution of 96 single base substitution types, 78 double bas substitutions types or indels. Mutational signatures can be extracted from your mutation count matrix, with non-negative matrix factorization (NMF). A critical parameter in NMF is the factorization rank, which is the number of mutational signatures. You can determine the optimal factorization rank using the NMF package (Gaujoux & Seoighe, 2010). As described in their paper:

"...a common way of deciding on the rank is to try different values, compute some quality measure of the results, and choose the best value according to this quality criteria. The most common approach is to choose the smallest rank for which cophenetic correlation coefficient starts decreasing. Another approach is to choose the rank for which the plot of the residual sum of squares (RSS) between the input matrix and its estimate shows an inflection point."

Lets start with the single base substitutions. First add a small psuedocount to your mutation count matrix, such that there are no rows where the sum of the row is zero:

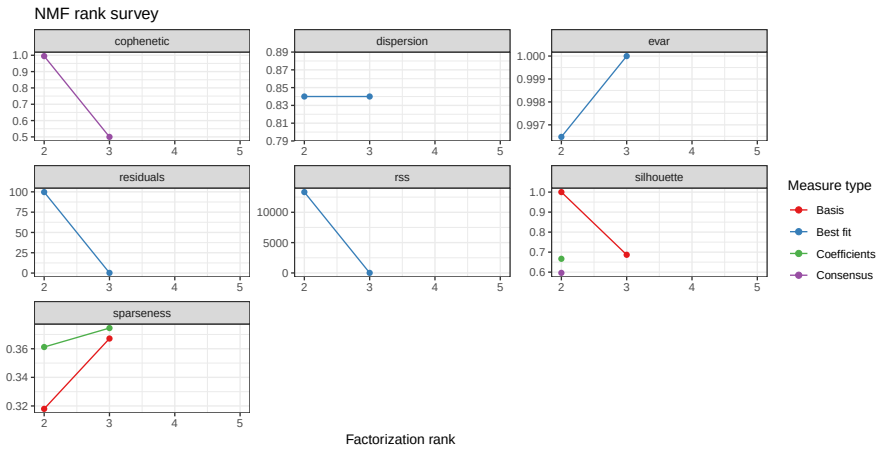
```
> mut_mat <- mut_matrix(vcf_list = vcfs_breast, ref_genome = ref_genome)
> mut_mat <- mut_mat + 0.0001
```

Use the NMF package to generate an estimate rank plot:

```
> library("NMF")
> estimate <- nmf(mut_mat, rank=2:5, method="brunet", nrun=10, seed=123456)
```

And plot it:

```
> plot(estimate)
```



Extract 2 mutational signatures from the mutation count matrix with `extract_signatures` (For larger datasets it is wise to perform more iterations by changing the nrun parameter to achieve stability and avoid local minima):

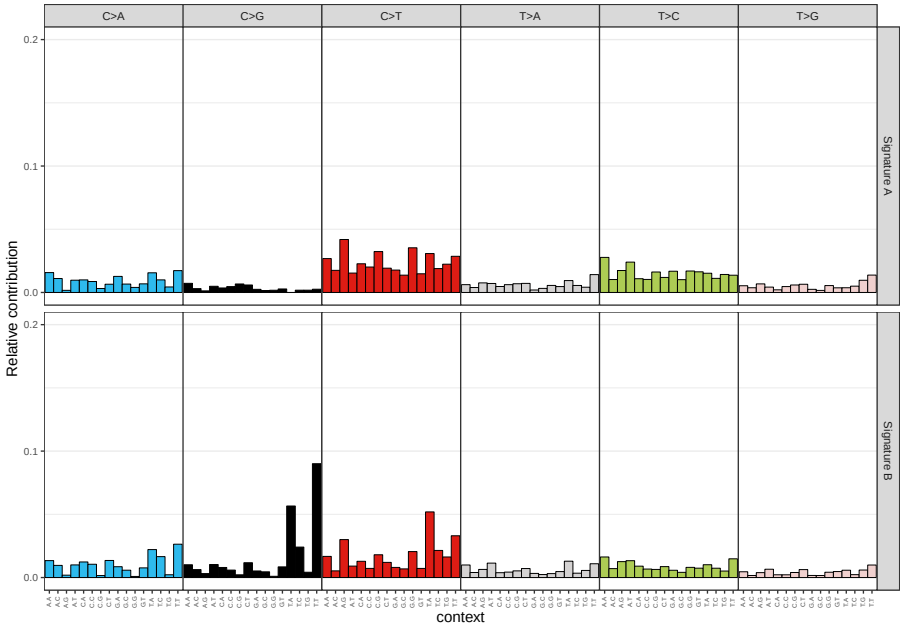
```
> nmf_res <- extract_signatures(mut_mat, rank = 2, nrun = 10)
```

Assign signature names:

```
> colnames(nmf_res$signatures) <- c("Signature A", "Signature B")
> rownames(nmf_res$contribution) <- c("Signature A", "Signature B")
```

Plot the 96-profile of the signatures:

```
> plot_profiles(nmf_res$signatures, condensed = TRUE)
```



In order to extract signatures for all mutation types at once, make a list of mutation matrices for each mutation type of the breast cancer samples:

```
> mut_mat <- mut_matrix(vcf_list = vcfs_breast, ref_genome = ref_genome, type = "all")
> mut_mat <- lapply(mut_mat, function(x) x + 0.0001)
```

Generate a estimate rank plot with the NMF package for each mutation type and find the best ranks. Extract then the signatures from the mutation matrices with `extract_signatures`. Use `type = "all"` to get all mutation types.

```
> nmf_res <- extract_signatures(mut_mat,
+                               rank = c("snv" = 2, "dbs" = 2, "indel" = 2),
+                               nrun = 10)
```

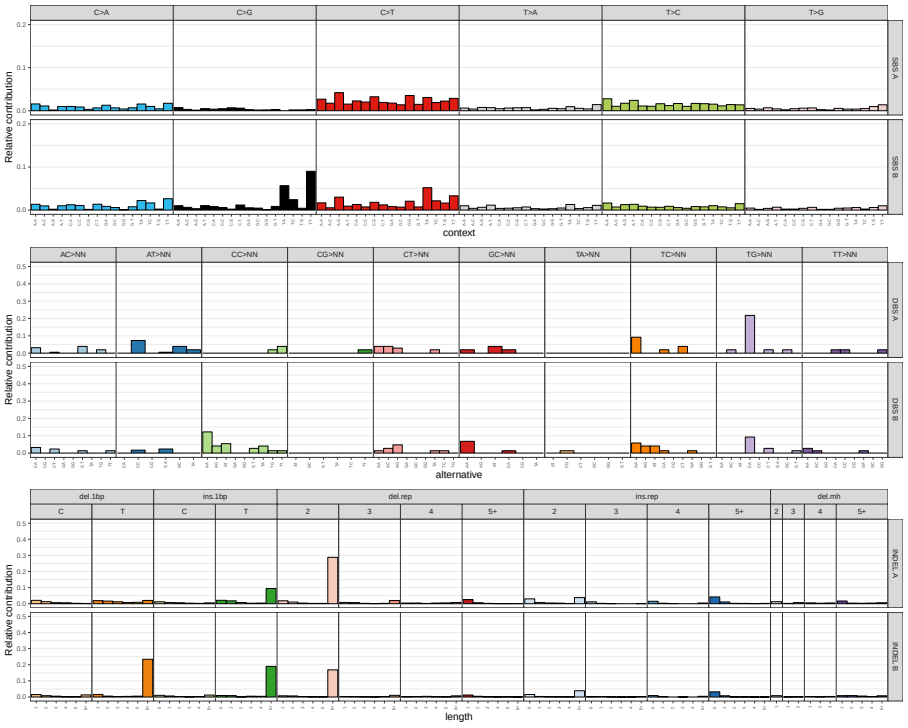
Assign signature names

```
> colnames(nmf_res$signatures$snv) <- c("SBS A", "SBS B")
> colnames(nmf_res$signatures$dbs) <- c("DBS A", "DBS B")
> colnames(nmf_res$signatures$indel) <- c("INDEL A", "INDEL B")
> rownames(nmf_res$contribution$snv) <- c("SBS A", "SBS B")
```

```
> rownames(nmf_res$contribution$dbs) <- c("DBS A", "DBS B")
> rownames(nmf_res$contribution$indel) <- c("INDEL A", "INDEL B")
```

Plot the profiles of the signatures:

```
> plot_profiles(nmf_res$signatures, condensed = TRUE)
```



Visualize the contribution of the SBS signatures in a barplot:

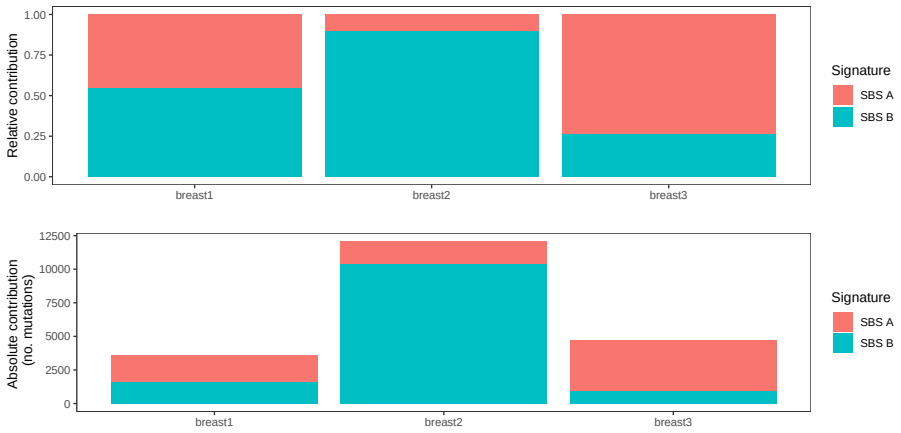
```
> pc1 <- plot_contribution(nmf_res$contribution, nmf_res$signature,
+                          type = "snv",
+                          mode = "relative")
```

Visualize the contribution of the signatures in absolute number of mutations:

```
> pc2 <- plot_contribution(nmf_res$contribution, nmf_res$signature,
+                          type = "snv",
+                          mode = "absolute")
```

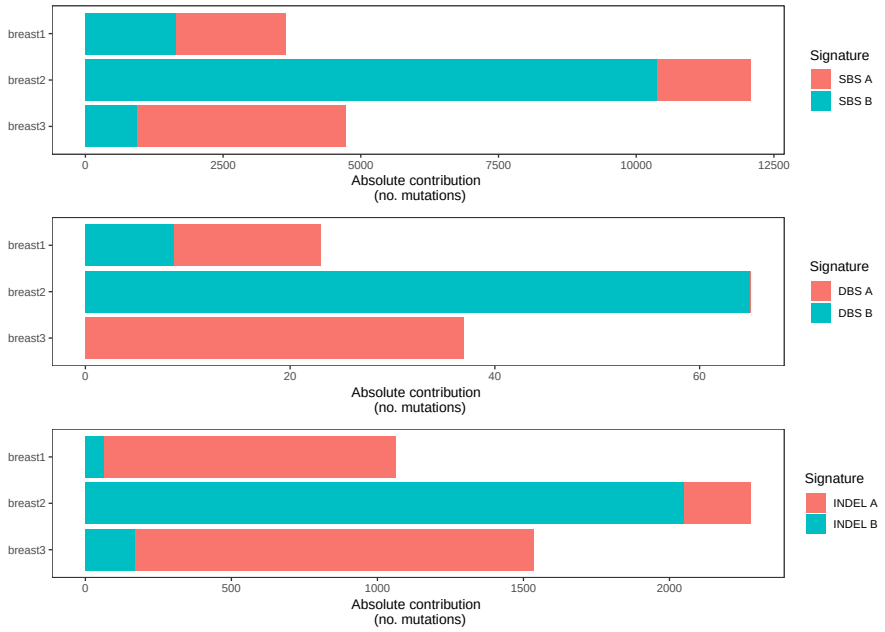
Combine the two plots:

```
> grid.arrange(pc1, pc2)
```



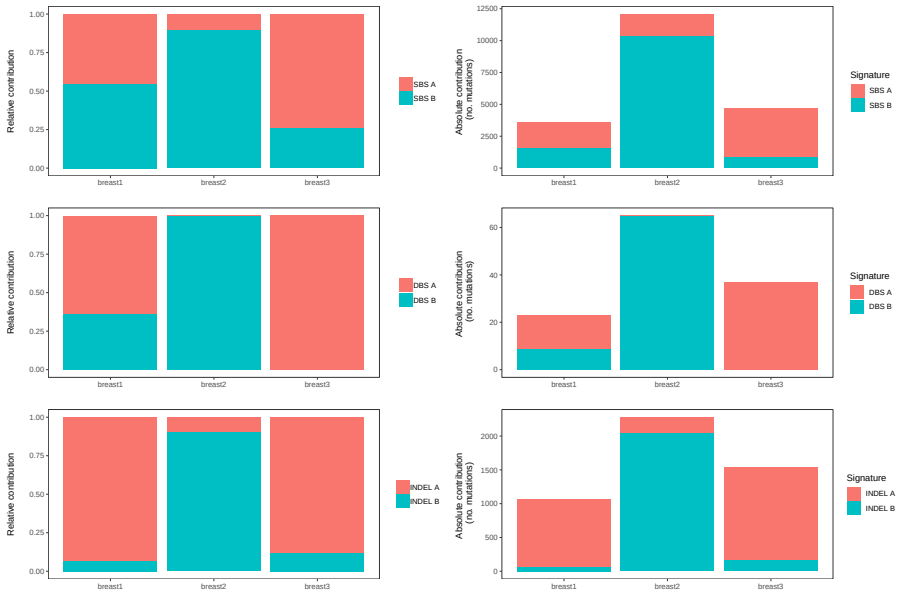
Flip X and Y coordinates:

```
> plot_contribution(nmf_res$contribution, nmf_res$signature,
+                          mode = "absolute", coord_flip = TRUE)
```



To visualize the contribution of the signatures for all mutation types in both relative and absolute number of mutations, set `type = "all"` and `mode = "both"`:

```
> plot_contribution(nmf_res$contribution, nmf_res$signature,
+                   type = "all", mode = "both")
```



The relative contribution of each signature for each sample can also be plotted as a heatmap with `plot_contribution_heatmap`, which might be easier to interpret and compare than stacked barplots. The samples can be hierarchically clustered based on their euclidean distance. The signatures can be plotted in a user-specified order.

Plot SBS signature contribution as a heatmap with sample clustering dendrogram and a specified signature order:

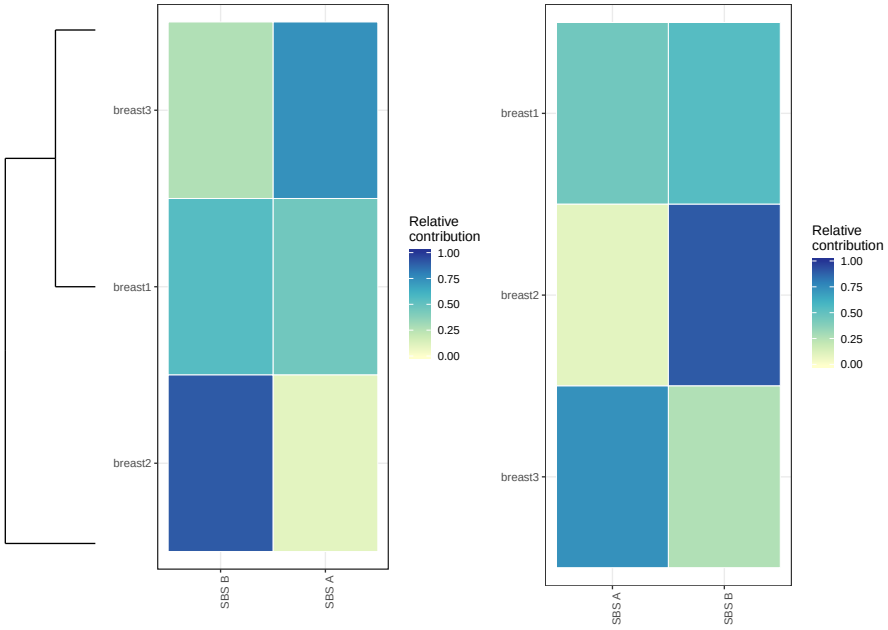
```
> pch1 <- plot_contribution_heatmap(nmf_res$contribution,
+                                   type = "snv",
+                                   sig_order = c("SBS B", "SBS A"))
```

Plot SBS signature contribution as a heatmap without sample clustering:

```
> pch2 <- plot_contribution_heatmap(nmf_res$contribution,
+                                   type = "snv",
+                                   cluster_samples=FALSE)
```

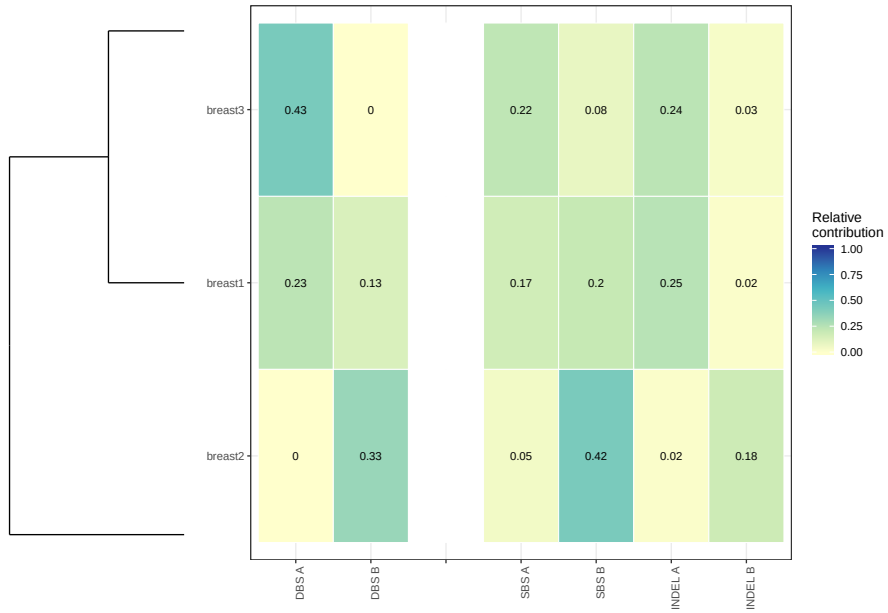
Combine the plots into one figure:

```
> grid.arrange(pch1, pch2, ncol = 2, widths = c(2,1.6))
```



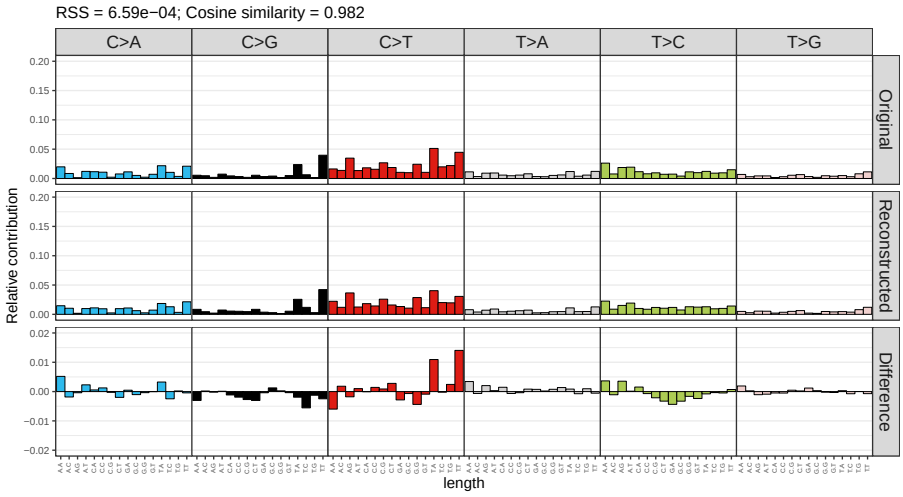
When plotting the signature contribution of multiple mutation types, it is possible to cluster on a specified mutation type. The mutation type(s) on which the data will be clustered, will show up at the left side of the heatmap. Plot the signature contribution, clustered by DBS signatures, by setting `cluster_mut_type = "dbs"`:

```
> plot_contribution_heatmap(nmf_res$contribution,
+                           cluster_mut_type = "dbs",
+                           plot_values = TRUE)
```



In order to see the performance of the NMF algorithm, a reconstruction of the count matrices are given by `extract_signatures`. Compare a reconstructed 96 mutational profile of SNVs with the original 96 mutational profile of SNVs:

```
> plot_compare_profiles(mut_mat$snv[,1],
+                       nmf_res$reconstructed$snv[,1],
+                       profile_names = c("Original", "Reconstructed"),
+                       condensed = TRUE)
```



4.2 Find optimal contribution of known signatures

4.2.1 COSMIC mutational signatures

Download mutational signatures from the COSMIC website. As there are multiple versions of the signatures, this vignette uses the signatures from COSMIC version 3 for SBS, DBS and indels. These signatures are available in numerical form from synapse.org ID syn12009743. Download here the referece whole genome signatures. Then load as follow:

```
> # Read the SBS signatures file
> # snv_signatures = read.csv("sigProfiler_SBS_signatures_v3_2019_05_22.csv")
> # Derive the 96 mutations
> # snv_signatures$MutationType = sprintf("%s[%s]%s",
>                                         # substr(snv_signatures$SubType, 1, 1),
>                                         # snv_signatures$Type,
>                                         # substr(snv_signatures$SubType, 3, 3))
>
> # Match the order of the mutation types to MutationalPatterns standard
> # new_order = match(row.names(mut_mat$snv), snv_signatures$MutationType)
> # Reorder cancer signatures dataframe
> # snv_signatures = snv_signatures[as.vector(new_order),]
> # Add trinucletiode changes names as row.names
> # row.names(snv_signatures) = snv_signatures$MutationType
> # Keep only 96 contributions of the signatures in matrix
```

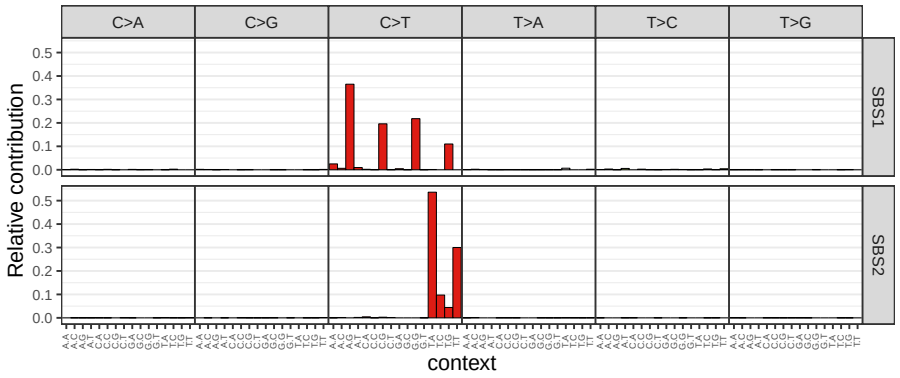
```
> # snv_signatures = as.matrix(snv_signatures[,3:69])
>
> # Read the DBS signatures file
> # dbs_signatures = read.csv("sigProfiler_DBS_signatures.csv")
> # Add mutation types as rownames
> # rownames(dbs_signatures) = dbs_signatures$Mutation.Type
> # Keep only 10 DBS signatures
> # dbs_signatures = as.matrix(dbs_signatures[,2:11])
>
> # Read the indel signatures file
> # indel_signatures = read.csv("sigProfiler_ID_signatures.csv")
> # Add indel context as rownames
> # rownames(indel_signatures) = MutationalPatterns:::INDEL_CONTEXT
> # Keep only the 17 indel signatures
> # indel_signatures = as.matrix(indel_signatures[,2:18])
>
> # Store all mutation types in one list
> # cosmic_signatures = list("snv" = snv_signatures,
> #                          "dbs" = dbs_signatures,
> #                          "indel" = indel_signatures)
>
> cosmic_signatures = readRDS(system.file("states/COSMIC_signatures.rds",
+                                         package = "MutationalPatterns"))
```

The SBS signatures from the COSMIC database include signatures which are probably because of sequencing artefacts. These signatures can better be removed before performing analyses.

```
> cosmic_signatures$snv = cosmic_signatures$snv[,-c(32,48,50:65)]
```

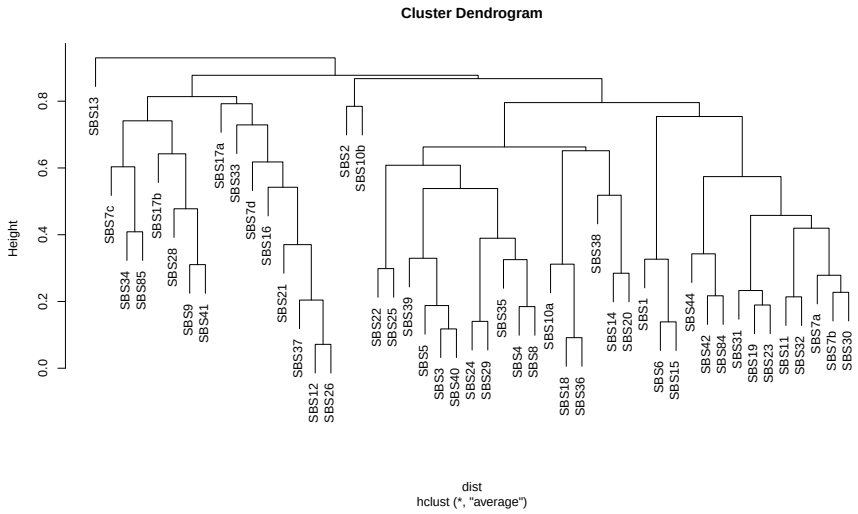
Plot mutational profile of the first two COSMIC SBS signatures:

```
> plot_profiles(cosmic_signatures$snv[,1:2], condensed = TRUE, ymax = "maximum")
```



Hierarchically cluster the COSMIC SBS signatures based on their similarity with average linkage:

```
> hclust_cosmic = cluster_signatures(cosmic_signatures$snv, method = "average")
> # store signatures in new order
> cosmic_order = colnames(cosmic_signatures$snv)[hclust_cosmic$order]
> plot(hclust_cosmic)
```



4.2.2 Similarity between mutational profiles and COSMIC signatures

The similarity between each mutational profile and each COSMIC signature, can be calculated with `cos_sim_matrix`, and visualized with `plot_cosine_heatmap`. The cosine similarity reflects how well each mutational profile can be explained by each signature individually. The advantage of this heatmap representation is that it shows in a glance the similarity in mutational profiles between samples, while at the same time providing information on which signatures are most prominent. The samples can be hierarchically clustered in `plot_cosine_heatmap`.

The cosine similarity between two mutational profiles/signatures can be calculated with `cos_sim`:

```
> cos_sim(mut_mat$snv[,1], cosmic_signatures$snv[,1])

[1] 0.4136337
```

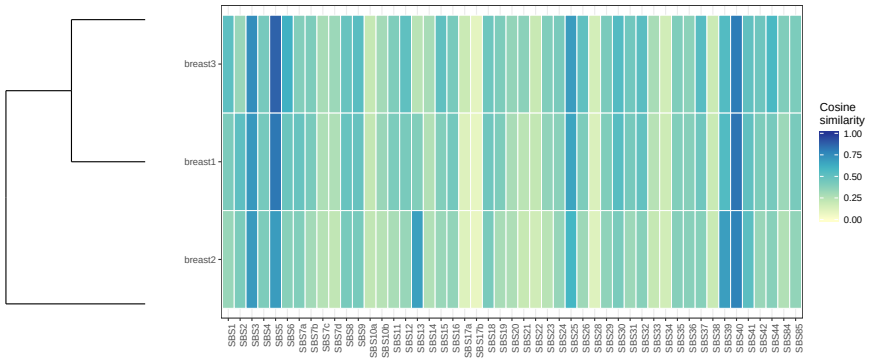
To do pairwise cosine similarity calculations of mutational profiles and COSMIC signatures, use the function `cos_sim_matrix`:

```
> cos_sim_samples_signatures = cos_sim_matrix(mut_mat, cosmic_signatures,
+                                             type = "all")
> # Print the first five signatures
> lapply(cos_sim_samples_signatures, function(x) x[,1:5])

$snv
             SBS1      SBS2      SBS3      SBS4      SBS5
breast1 0.4136337 0.5207366 0.7070212 0.4367681 0.8196882
breast2 0.3356698 0.4105033 0.6998797 0.4070196 0.6949269
breast3 0.5234627 0.3336801 0.7427320 0.4337026 0.8777017
```

```
$dbs
               DBS1        DBS2      DBS3      DBS4       DBS5
breast1 0.006468408 0.354192648 0.2450052 0.1348871 0.03253983
breast2 0.085060850 0.532678429 0.2264256 0.4596892 0.18115234
breast3 0.188296917 0.005248677 0.1683697 0.3542400 0.27796420

$indel
              ID1       ID2       ID3        ID4       ID5
breast1 0.2257081 0.1264387 0.1174937 0.07768175 0.2280121
breast2 0.5292935 0.6346073 0.1261506 0.05322776 0.2176745
breast3 0.4151302 0.1346788 0.1416322 0.09373001 0.2578521
```

Plot the cosine similarity heatmap of the SBS signatures:

```
> plot_cosine_heatmap(cos_sim_samples_signatures$snv,
+                     cluster_rows = TRUE)
```



4.2.3 Find optimal contribution of COSMIC signatures to reconstruct mutational profiles

In addition to *de novo* extraction of signatures, the contribution of any set of signatures to the mutational profile of a sample can be quantified. This unique feature is specifically useful for mutational signature analyses of small cohorts or individual samples, but also to relate own findings to known signatures and published findings. The `fit_to_signatures` function has two options to find the optimal linear combination of mutational signatures that most closely reconstructs the mutation matrix: solving a non-negative least-squares constraints problem and performing a golden ratio search (as implemented in the deconstructSigs package from Rosenthal et al. (Rosenthal, McGranahan, Herrero, Taylor, & Swanton, 2016)). The default option is the non-negative least-squares problem.

First get new mutation matrices, without the 0.001 used by the NMF estimation:

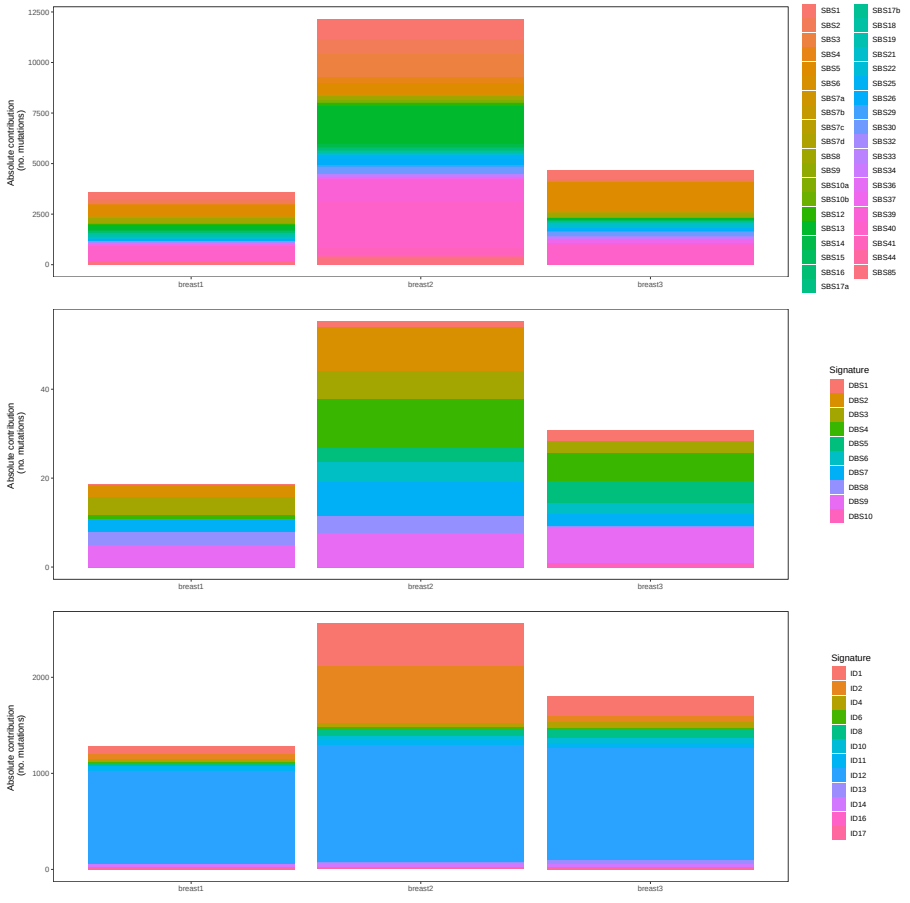
```
> mut_mat <- mut_matrix(vcf_list = vcfs_breast, ref_genome, type = "all")
```

Fit mutation matrices to the COSMIC signatures:

```
> fit_res <- fit_to_signatures(mut_mat, cosmic_signatures)
```

Plot the optimal contribution of the COSMIC signatures in each sample as a stacked barplot.

```
> # Select signatures with some contribution
> fit_res$contribution$snv <- fit_res$contribution$snv[
+   which(rowSums(fit_res$contribution$snv) > 10),]
> fit_res$contribution$dbs <- fit_res$contribution$dbs[
+   which(rowSums(fit_res$contribution$dbs) > 0.1),]
> fit_res$contribution$indel <- fit_res$contribution$indel[
+   which(rowSums(fit_res$contribution$indel) > 10),]
> # Plot contribution barplot
> plot_contribution(fit_res$contribution,
+                   cosmic_signatures,
+                   coord_flip = FALSE,
+                   mode = "absolute")
```



Results of the golden ratio search algorithm are only relative, so fit the mutation matrix with the golden ratio search and plot results from both methods in relative contribution for the point mutations:

```
> fit_res_grs <- fit_to_signatures(mut_mat, cosmic_signatures, type = "snv",
+                                  method = "golden-ratio-search")
> # Select signatures with some contribution
> select_grs <- which(rowSums(fit_res_grs$contribution) > 0.06)
```

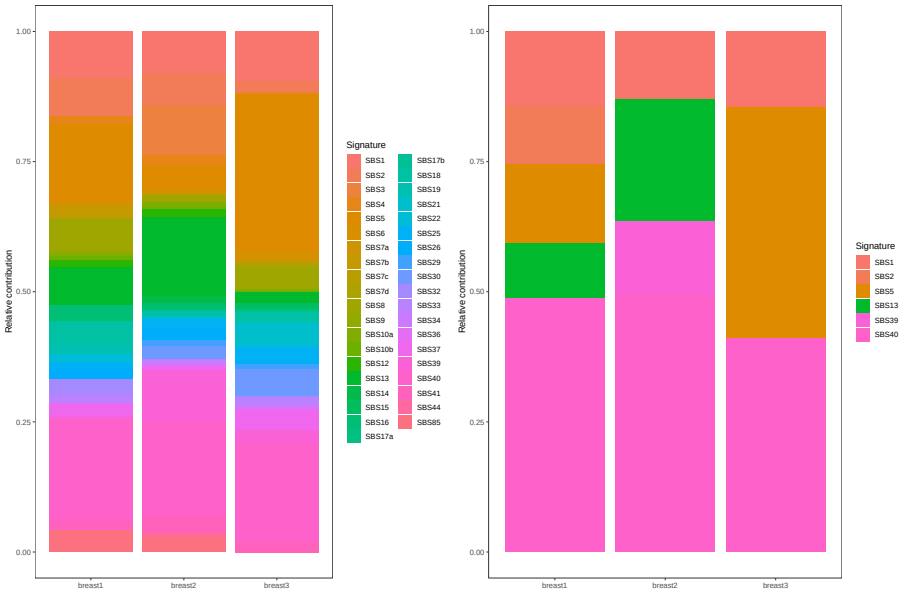
In order to match colors when `plot_contribution` is run for both the non-negative least squares problem and the golden ratio search, make a palette of colors with the `default_colors_ggplot` function:

```
> colorvector <- default_colors_ggplot(ncol(cosmic_signatures$snv))
```

Then plot the results of both algorithms:

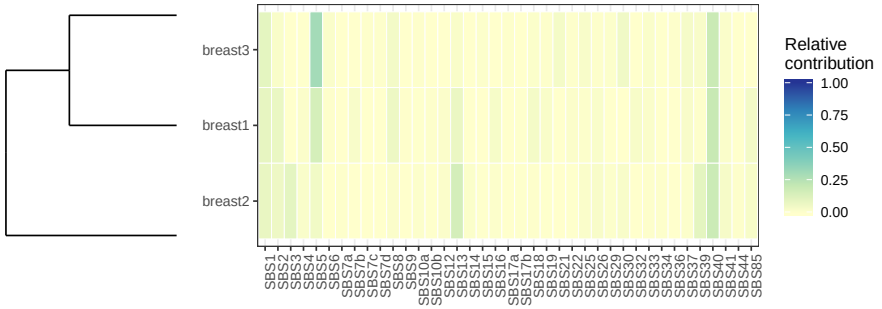
```
> # Plot relative contribution from non-negative least squares
> select = match(rownames(fit_res$contribution$snv), colnames(cosmic_signatures$snv))
> pc1 <- plot_contribution(fit_res$contribution,
+                     cosmic_signatures$snv,
+                     coord_flip = FALSE,
+                     type = "snv",
+                     mode = "relative",
+                     palette = list("snv" = colorvector[select]))
> # Plot relative contribution from golden ratio search
> pc2 <- plot_contribution(fit_res_grs$contribution[select_grs,],
+                          cosmic_signatures$snv[,select_grs],
+                          coord_flip = FALSE,
+                          mode = "relative",
+                          palette = list("snv" = colorvector[select_grs]))
```

Combine the two plots in one figure:



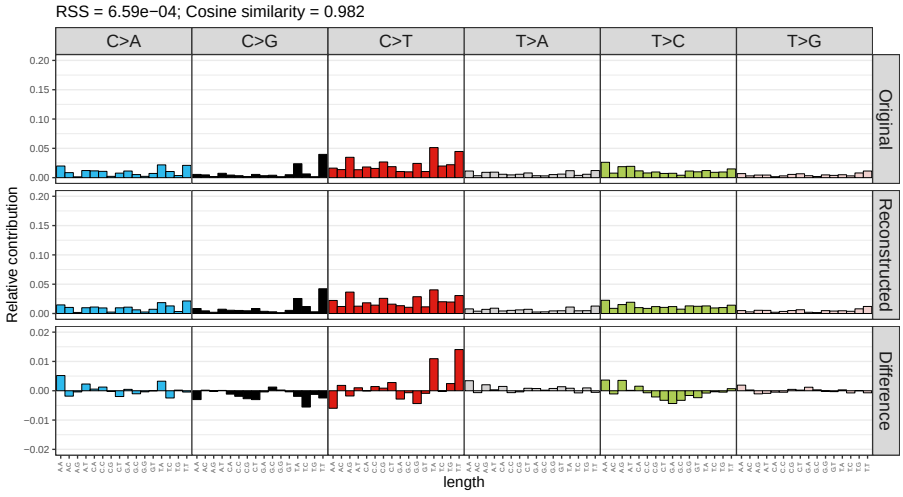
The relative contributions of signatures to samples can be plotted as a heatmap. Plot the contribution heatmap of the SBS signatures:

```
> plot_contribution_heatmap(fit_res$contribution$snv,
+                           cluster_samples = TRUE,
+                           method = "complete")
```



A quality control of the fitted signatures is to compare the reconstructed mutational profiles with the orignals. This can be done with the function `plot_compare_profiles`. Compare the reconstructed mutational profile of indels of sample 1 with its original mutational profile of indels:

```
> plot_compare_profiles(mut_mat$indel[,1], fit_res$reconstructed$indel[,1],
+                       profile_names = c("Original", "Reconstructed"),
+                       condensed = TRUE)
```



Calculate the cosine similarity between all original and reconstructed mutational profiles with `cos_sim_matrix`:

```
> # calculate all pairwise cosine similarities
> cos_sim_ori_rec <- cos_sim_matrix(mut_mat, fit_res$reconstructed, type = "all")
> # extract cosine similarities per sample between original and reconstructed
```

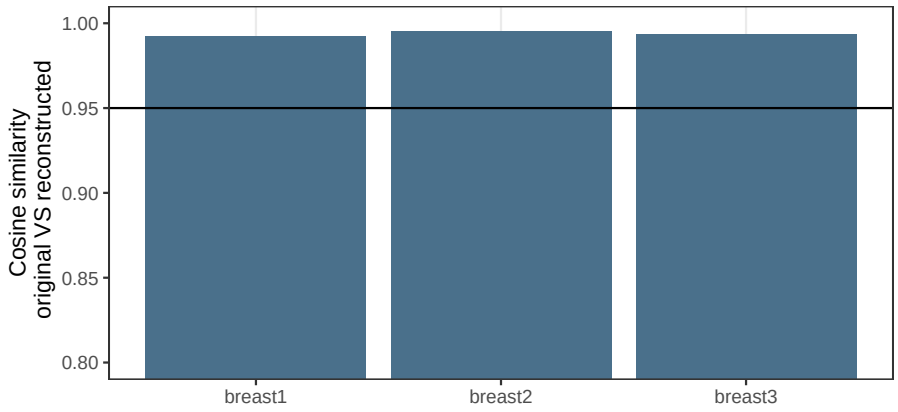
```
> cos_sim_ori_rec <- lapply(cos_sim_ori_rec, function(x) as.data.frame(diag(x)))
```

We can use ggplot to make a barplot of the cosine similarities between the original and reconstructed mutational profile of each sample. This clearly shows how well each mutational profile can be reconstructed with the COSMIC mutational signatures. Two identical profiles have a cosine similarity of 1. The lower the cosine similarity between original and reconstructed, the less well the original mutational profile can be reconstructed with the COSMIC signatures. You could use, for example, cosine similarity of 0.95 as a cutoff.

```
> # Adjust data frame for plotting with gpplot
> for (i in 1:length(cos_sim_ori_rec)){
+   colnames(cos_sim_ori_rec[[i]]) = "cos_sim"
+   cos_sim_ori_rec[[i]]$sample = row.names(cos_sim_ori_rec[[i]])
+ }
```

Plot the cosine similarities for the SBS signatures:

```
> # Load ggplot2
> library(ggplot2)
> # Make barplot
> ggplot(cos_sim_ori_rec$snv, aes(y=cos_sim, x=sample)) +
+   geom_bar(stat="identity", fill = "skyblue4") +
+   coord_cartesian(ylim=c(0.8, 1)) +
+   # coord_flip(ylim=c(0.8,1)) +
+   ylab("Cosine similarity\n original VS reconstructed") +
+   xlab("") +
+   # Reverse order of the samples such that first is up
+   # xlim(rev(levels(factor(cos_sim_ori_rec$sample)))) +
+   theme_bw() +
+   theme(panel.grid.minor.y=element_blank(),
+         panel.grid.major.y=element_blank()) +
+   # Add cut.off line
+   geom_hline(aes(yintercept=.95))
```



5 Strand bias analyses

5.1 Transcriptional strand bias analysis

For the mutations within genes it can be determined whether the mutation is on the transcribed or non-transcribed strand, which can be used to evaluate the involvement of transcription-coupled repair. To this end, it is determined whether the "C" or "T" base (since by convention we regard base substitutions as C>X or T>X) are on the same strand as the gene definition. Single base substitions on the same strand as the gene definitions are considered "untranscribed", and on the opposite strand of gene bodies as "transcribed", since the gene definitions report the coding or sense strand, which is untranscribed. No strand information is reported for base substitution that overlap with more than one gene body on different strands.

Alike the single base substitutions, double base substitutions are converted to defined set of double bases. These bases are either on the same strand as a gene definition, consider them "untranscribed", or on the other strand, consider them "transcribed". Indels do not have such a conversion, therefore losing strand information based on mutations.

Get gene definitions for your reference genome:

```
> # For example get known genes table from UCSC for hg19 using
> # biocLite("TxDb.Hsapiens.UCSC.hg19.knownGene")
> library("TxDb.Hsapiens.UCSC.hg19.knownGene")
> genes_hg19 <- genes(TxDb.Hsapiens.UCSC.hg19.knownGene)
> genes_hg19

GRanges object with 23056 ranges and 1 metadata column:
        seqnames                ranges strand |     gene_id
           <Rle>             <IRanges>  <Rle> | <character>
      1    chr19 [ 58858172,  58874214]      - |           1
     10     chr8 [ 18248755,  18258723]      + |          10
    100    chr20 [ 43248163,  43280376]      - |         100
   1000    chr18 [ 25530930,  25757445]      - |        1000
  10000     chr1 [243651535, 244006886]      - |       10000
    ...      ...                   ...    ... .         ...
   9991     chr9 [114979995, 115095944]      - |        9991
   9992    chr21 [ 35736323,  35743440]      + |        9992
   9993    chr22 [ 19023795,  19109967]      - |        9993
   9994     chr6 [ 90539619,  90584155]      + |        9994
   9997    chr22 [ 50961997,  50964905]      - |        9997
  -------
  seqinfo: 93 sequences (1 circular) from hg19 genome
```

Get transcriptional strand information for all SBS and DBS positions in the first VCF object with `mut_strand`. This function returns "-" for positions outside gene bodies, and positions that overlap with more than one gene on different strands. Use the vcfs with the different tissue types:

```
> strand = mut_strand(vcfs_tissues[[1]], genes_hg19)
> head(strand, 10)

 [1] -             -             -             transcribed   untranscribed -
 [7] transcribed   -             untranscribed untranscribed
```

```
Levels: untranscribed transcribed -
```

Make mutation count matrix with transcriptional strand information (96 trinucleotides * 2 strands = 192 features for SBS and 78 substitutions * 2 strands = 156 features for DBS). NB: only those mutations that are located within gene bodies are counted.

```
> mut_mat_s <- mut_matrix_stranded(vcfs_tissues, ref_genome, genes_hg19)
> mut_mat_s[1:5,1:5]

                     colon1 colon2 colon3 intestine1 intestine2
A[C>A]A-untranscribed      0      0      0          0          4
A[C>A]A-transcribed        1      1      2          4          3
A[C>A]C-untranscribed      0      0      1          1          1
A[C>A]C-transcribed        0      0      0          0          1
A[C>A]G-untranscribed      1      0      0          0          0
```

Count the number of mutations on each strand, per tissue, per mutation type:

```
> strand_counts <- strand_occurrences(mut_mat_s, by=tissue)
> head(strand_counts, 10)

   group mutation type        strand no_mutations relative_contribution
1  colon      snv  C>A   transcribed           32            0.07289294
4  colon      snv  C>A untranscribed           23            0.05239180
7  colon      snv  C>G   transcribed           11            0.02505695
10 colon      snv  C>G untranscribed           10            0.02277904
13 colon      snv  C>T   transcribed          135            0.30751708
16 colon      snv  C>T untranscribed          115            0.26195900
19 colon      snv  T>A   transcribed           12            0.02733485
22 colon      snv  T>A untranscribed            9            0.02050114
25 colon      snv  T>C   transcribed           36            0.08200456
28 colon      snv  T>C untranscribed           32            0.07289294
```

Perform Poisson test for strand asymmetry significance testing:

```
> strand_bias <- strand_bias_test(strand_counts)
> strand_bias

       group mutation type transcribed untranscribed total     ratio  p_poisson significant
1      colon      snv  C>A          32            23    55 1.3913043 0.28060972
2      colon      snv  C>G          11            10    21 1.1000000 1.00000000
3      colon      snv  C>T         135           115   250 1.1739130 0.22942486
4      colon      snv  T>A          12             9    21 1.3333333 0.66362381
5      colon      snv  T>C          36            32    68 1.1250000 0.71630076
6      colon      snv  T>G          15             9    24 1.6666667 0.30745625
7  intestine      snv  C>A          34            27    61 1.2592593 0.44262600
8  intestine      snv  C>G          18            21    39 0.8571429 0.74925862
9  intestine      snv  C>T         144           129   273 1.1162791 0.39685899
10 intestine      snv  T>A          23            18    41 1.2777778 0.53270926
11 intestine      snv  T>C          52            38    90 1.3684211 0.17024240
12 intestine      snv  T>G          10            10    20 1.0000000 1.00000000
13     liver      snv  C>A          45            44    89 1.0227273 1.00000000
14     liver      snv  C>G          19            34    53 0.5588235 0.05343881
15     liver      snv  C>T          87            82   169 1.0609756 0.75842199
```

```
16     liver      snv  T>A          36            23     59 1.5652174 0.11747735
17     liver      snv  T>C          75            52    127 1.4423077 0.05048701
18     liver      snv  T>G          23            43     66 0.5348837 0.01865726   *
```

Plot the mutation spectrum with strand distinction:

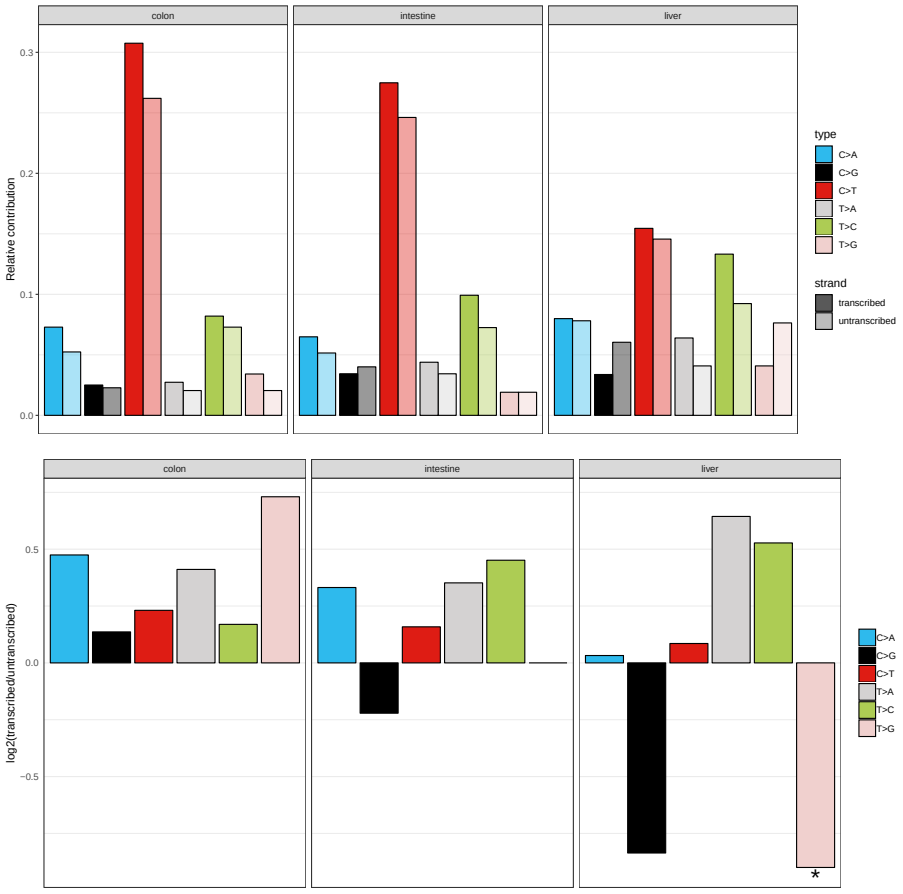
```
> ps1 <- plot_strand(strand_counts, mode = "relative")
```

Plot the effect size (log2(untranscribed/transcribed) of the strand bias. Asteriks indicate significant strand bias.

```
> ps2 <- plot_strand_bias(strand_bias)
```

Combine the plots into one figure:

```
> grid.arrange(ps1, ps2)
```



5.2 Replicative strand bias analysis

The involvement of replication-associated mechanisms can be evaluated by testing for a mutational bias between the leading and lagging strand. The replication strand is dependent on the locations of replication origins from which DNA replication is fired. However, replication timing is dynamic and cell-type specific, which makes replication strand determination less straightforward than transcriptional strand bias analysis. Replication timing profiles can be generated with Repli-Seq experiments. Once the replication direction is defined, a strand asymmetry analysis can be performed similarly as the transcription strand bias analysis.

Read example bed file provided with the package with replication direction annotation:

```
> repli_file = system.file("extdata/ReplicationDirectionRegions.bed",
+                          package = "MutationalPatterns")
> repli_strand = read.table(repli_file, header = TRUE)
> # Store in GRanges object
> repli_strand_granges = GRanges(seqnames = repli_strand$Chr,
+   ranges = IRanges(start = repli_strand$Start + 1,
+                    end = repli_strand$Stop),
+   strand_info = factor(repli_strand$Class))
> # UCSC seqlevelsstyle
> seqlevelsStyle(repli_strand_granges) = "UCSC"
> repli_strand_granges
```

```
GRanges object with 1993 ranges and 1 metadata column:
         seqnames               ranges strand | strand_info
            <Rle>            <IRanges>  <Rle> |    <factor>
     [1]     chr1   [2133001, 3089000]      * |       right
     [2]     chr1   [3089001, 3497000]      * |        left
     [3]     chr1   [3497001, 4722000]      * |       right
     [4]     chr1   [5223001, 6428000]      * |        left
     [5]     chr1   [6428001, 7324000]      * |       right
     ...      ...                 ...    ... .         ...
  [1989]     chrY [23997001, 24424000]      * |       right
  [1990]     chrY [24424001, 28636000]      * |        left
  [1991]     chrY [28636001, 28686000]      * |       right
  [1992]     chrY [28686001, 28760000]      * |        left
  [1993]     chrY [28760001, 28842000]      * |       right
  -------
  seqinfo: 24 sequences from an unspecified genome; no seqlengths
```

The GRanges object should have a "strand_info" metadata column, which contains only two different annotations, e.g. "left" and "right", or "leading" and "lagging". The genomic ranges cannot overlap, to allow only one annotation per location.

Get replicative strand information for all positions in the first VCF object. No strand information "-" is returned for base substitutions in unannotated genomic regions. Indels can also be tested for replication strand bias, since the strand information is not based on conversion of mutations.

```
> strand_rep <- mut_strand(vcfs_tissues[[1]], repli_strand_granges,
+                          mode = "replication")
> head(strand_rep, 10)
```

```
 [1] -     left  left  left  right left  -     -     -     left
Levels: left right -
```

Make mutation count matrices with transcriptional strand information.

```
> mut_mat_s_rep <- mut_matrix_stranded(vcfs_tissues, ref_genome, repli_strand_granges,
+                                      mode = "replication")
> mut_mat_s_rep[1:5, 1:5]
```

```
              colon1 colon2 colon3 intestine1 intestine2
A[C>A]A-left       2      1      0          0          3
A[C>A]A-right      0      3      2          2          5
A[C>A]C-left       0      1      1          0          1
A[C>A]C-right      0      0      1          0          3
A[C>A]G-left       0      0      1          1          0
```

The levels of the "strand_info" metadata in the GRanges object determines the order in which the strands are reported in the mutation matrix that is returned by `mut_matrix_stranded`, so if you want to count right before left, you can specify this, before you run `mut_matrix_stranded`:

```
> repli_strand_granges$strand_info <- factor(repli_strand_granges$strand_info,
+                                            levels = c("right", "left"))
> mut_mat_s_rep2 <- mut_matrix_stranded(vcfs_tissues, ref_genome, repli_strand_granges,
+                                       mode = "replication")
> mut_mat_s_rep2[1:5, 1:5]
```

```
              colon1 colon2 colon3 intestine1 intestine2
A[C>A]A-right      0      3      2          2          5
A[C>A]A-left       2      1      0          0          3
A[C>A]C-right      0      0      1          0          3
A[C>A]C-left       0      1      1          0          1
A[C>A]G-right      0      1      1          0          1
```

Count the number of mutations on each strand, per tissue, per mutation type:

```
> strand_counts_rep <- strand_occurrences(mut_mat_s_rep, by=tissue)
> head(strand_counts_rep)
```

```
   group mutation type strand no_mutations relative_contribution
1  colon      snv  C>A   left           28            0.05490196
4  colon      snv  C>A  right           42            0.08235294
7  colon      snv  C>G   left           12            0.02352941
10 colon      snv  C>G  right           12            0.02352941
13 colon      snv  C>T   left          157            0.30784314
16 colon      snv  C>T  right          128            0.25098039
```

Perform Poisson test for strand asymmetry significance testing:

```
> strand_bias_rep <- strand_bias_test(strand_counts_rep)
> strand_bias_rep
```

```
       group mutation type left right total     ratio  p_poisson significant
1      colon      snv  C>A   28    42    70 0.6666667 0.11960934
2      colon      snv  C>G   12    12    24 1.0000000 1.00000000
```

```
3      colon      snv  C>T  157   128   285 1.2265625 0.09702977
4      colon      snv  T>A   12    10    22 1.2000000 0.83181190
5      colon      snv  T>C   41    41    82 1.0000000 1.00000000
6      colon      snv  T>G   16    11    27 1.4545455 0.44206834
7  intestine      snv  C>A   31    33    64 0.9393939 0.90065325
8  intestine      snv  C>G   19    11    30 1.7272727 0.20048842
9  intestine      snv  C>T  146   162   308 0.9012346 0.39274995
10 intestine      snv  T>A   21    15    36 1.4000000 0.40503225
11 intestine      snv  T>C   45    34    79 1.3235294 0.26042553
12 intestine      snv  T>G   10    11    21 0.9090909 1.00000000
13     liver      snv  C>A   47    51    98 0.9215686 0.76203622
14     liver      snv  C>G   34    33    67 1.0303030 1.00000000
15     liver      snv  C>T  107    98   205 1.0918367 0.57644403
16     liver      snv  T>A   24    31    55 0.7741935 0.41875419
17     liver      snv  T>C   75    63   138 1.1904762 0.34911517
18     liver      snv  T>G   29    34    63 0.8529412 0.61465502
```

Plot the mutation spectrum with strand distinction:

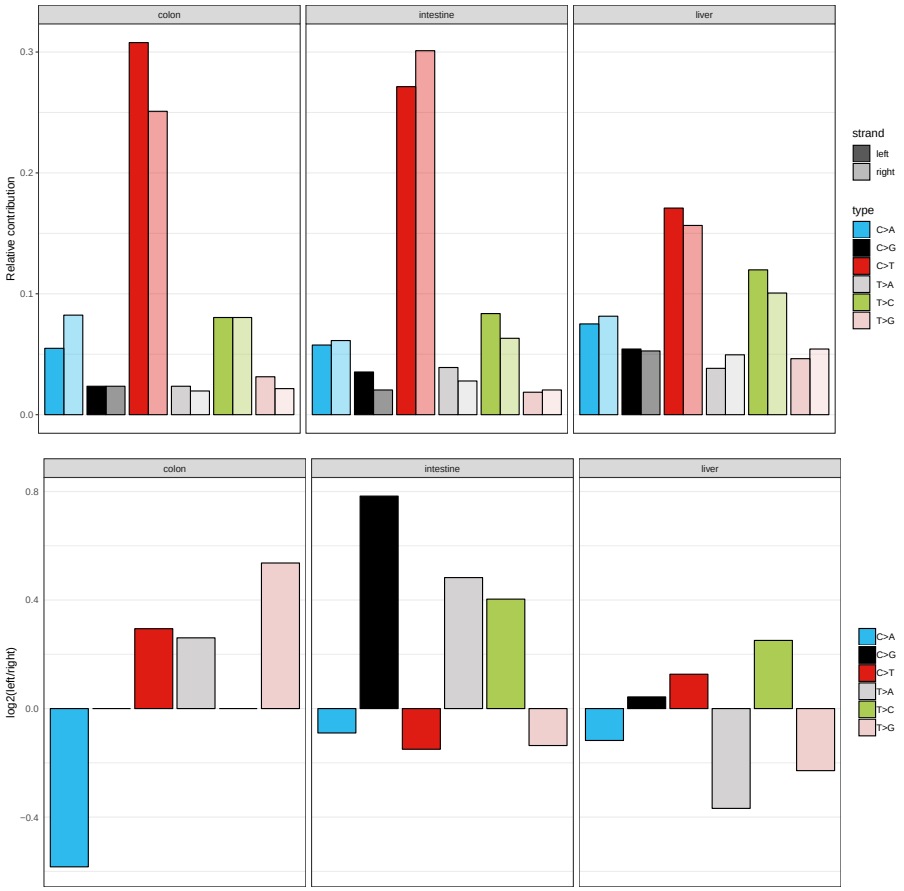
```
> ps1 <- plot_strand(strand_counts_rep, mode = "relative")
```

Plot the effect size (log2(untranscribed/transcribed) of the strand bias. Asteriks indicate significant strand bias.

```
> ps2 <- plot_strand_bias(strand_bias_rep)
```

Combine the plots into one figure:

```
> grid.arrange(ps1, ps2)
```



5.3 Replicative strand bias for DBS and indel

The previous analysis for replicative strand bias can also be performed for double base substitutions and indels. Use the breast cancer vcfs for these mutation types:

```
> strand_rep <- mut_strand(vcfs_breast[[1]], repli_strand_granges,
+                          mode = "replication", type = "all")
> lapply(strand_rep, head, 10)

$snv
 [1] -     -     -     right right right left  -     left  right
Levels: right left -

$dbs
 [1] -     -     left  -     -     -     -     -     right left
Levels: right left -

$indel
 [1] -     left  left  left  -     -     -     left  -     right
Levels: right left -
```

Make mutation count matrices with transcriptional strand information.

```
> mut_mat_s_rep <- mut_matrix_stranded(vcfs_breast, ref_genome, repli_strand_granges,
+                                      mode = "replication", type = "all")
> lapply(mut_mat_s_rep, function(x) x[1:5,])

$snv
              breast1 breast2 breast3
A[C>A]A-right      11      34      14
A[C>A]A-left        9      20       8
A[C>A]C-right       6      29       9
A[C>A]C-left       10      23      17
A[C>A]G-right       0       4       5

$dbs
            breast1 breast2 breast3
AC>CA-right       1       0       1
AC>CA-left        0       1       0
AC>CG-right       0       0       0
AC>CG-left        0       0       0
AC>CT-right       0       0       0

$indel
                              breast1 breast2 breast3
del.1bp.homopol.C.len.1-right       5       6       6
del.1bp.homopol.C.len.1-left        6      11      10
del.1bp.homopol.C.len.2-right       1       2       6
del.1bp.homopol.C.len.2-left        5       7       4
del.1bp.homopol.C.len.3-right       1       5       2
```

Count the number of mutations on each strand, per mutation type:

```
> strand_counts_rep <- strand_occurrences(mut_mat_s_rep)
> lapply(strand_counts_rep, head)

$snv
  group mutation type strand no_mutations relative_contribution
1   all      snv  C>A   left          591            0.07227590
2   all      snv  C>A  right          574            0.07019689
3   all      snv  C>G   left          715            0.08744038
4   all      snv  C>G  right          716            0.08756268
5   all      snv  C>T   left         1414            0.17292406
6   all      snv  C>T  right         1379            0.16864376

$dbs
  group mutation type strand no_mutations relative_contribution
1   all      dbs   AC   left            2            0.05128205
2   all      dbs   AC  right            2            0.05128205
3   all      dbs   AT   left            0            0.00000000
4   all      dbs   AT  right            2            0.05128205
5   all      dbs   CC   left            4            0.10256410
6   all      dbs   CC  right            3            0.07692308

$indel
```

```
  group mutation              type strand no_mutations relative_contribution
1   all    indel del.1bp.homopol.C   left           65           0.029802843
2   all    indel del.1bp.homopol.C  right           48           0.022008253
3   all    indel del.1bp.homopol.T   left          151           0.069234296
4   all    indel del.1bp.homopol.T  right          168           0.077028886
5   all    indel      del.mh.len.2   left            9           0.004126547
6   all    indel      del.mh.len.2  right           12           0.005502063
```

Perform Poisson test for strand asymmetry significance testing:

```
> strand_bias_rep <- strand_bias_test(strand_counts_rep)
> lapply(strand_bias_rep, head)

$snv
  group mutation type left right total     ratio p_poisson significant
1   all      snv  C>A  591   574  1165 1.0296167 0.6392549
2   all      snv  C>G  715   716  1431 0.9986034 1.0000000
3   all      snv  C>T 1414  1379  2793 1.0253807 0.5200080
4   all      snv  T>A  332   346   678 0.9595376 0.6176274
5   all      snv  T>C  777   733  1510 1.0600273 0.2684728
6   all      snv  T>G  302   298   600 1.0134228 0.9025366

$dbs
  group mutation type left right total     ratio p_poisson significant
1   all      dbs   AC    2     2     4 1.0000000 1.0000000
2   all      dbs   AT    0     2     2 0.0000000 0.5000000
3   all      dbs   CC    4     3     7 1.3333333 1.0000000
4   all      dbs   CG    0     1     1 0.0000000 1.0000000
5   all      dbs   CT    2     6     8 0.3333333 0.2890625
6   all      dbs   GC    1     2     3 0.5000000 1.0000000

$indel
  group mutation              type left right total     ratio p_poisson significant
1   all    indel del.1bp.homopol.C   65    48   113 1.3541667 0.1319243
2   all    indel del.1bp.homopol.T  151   168   319 0.8988095 0.3703699
3   all    indel      del.mh.len.2    9    12    21 0.7500000 0.6636238
4   all    indel      del.mh.len.3    4     2     6 2.0000000 0.6875000
5   all    indel      del.mh.len.4   18    11    29 1.6363636 0.2649309
6   all    indel     del.mh.len.5+   43    39    82 1.1025641 0.7406528
```

Plot the mutation spectrum with strand distinction:

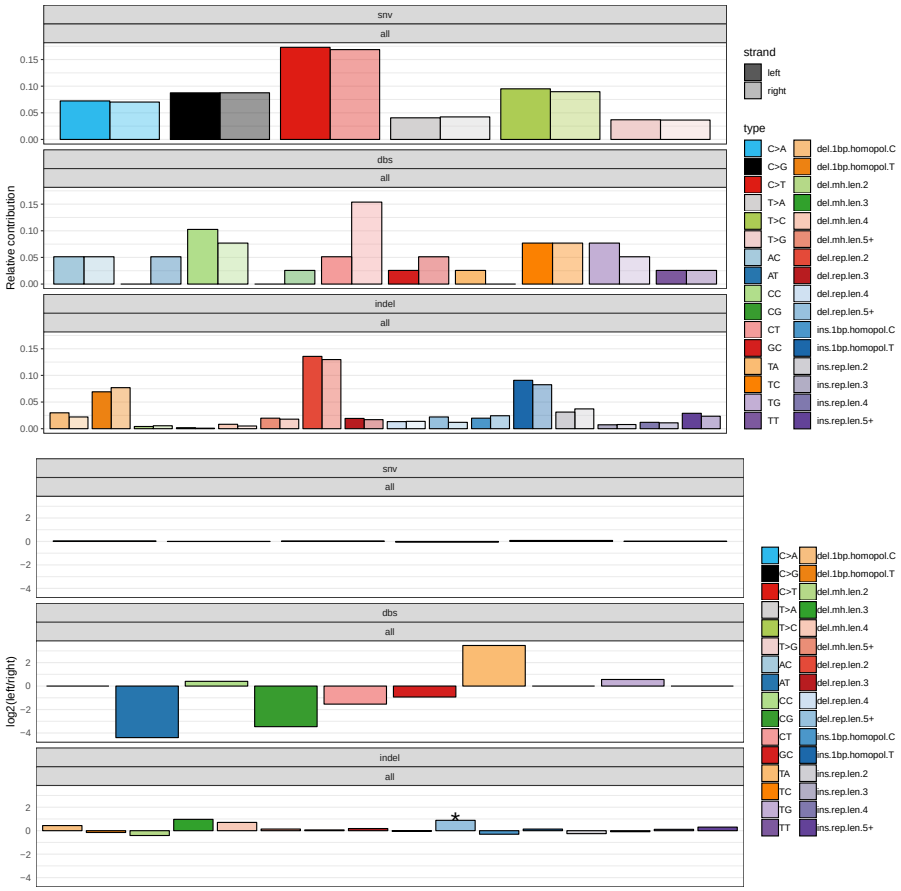
```
> ps1 <- plot_strand(strand_counts_rep, mode = "relative")
```

Plot the effect size (log2(untranscribed/transcribed) of the strand bias. Asteriks indicate significant strand bias.

```
> ps2 <- plot_strand_bias(strand_bias_rep)
```

Combine the plots into one figure:

```
> grid.arrange(ps1, ps2)
```



5.4 Extract signatures with strand bias

Extract 2 signatures for each mutation type from mutation count matrix with strand features:

```
> nmf_res_strand <- extract_signatures(mut_mat_s_rep, rank = 2, nrun = 1)
> # Provide signature names
> colnames(nmf_res_strand$signatures$snv) <- c("SBS A", "SBS B")
> colnames(nmf_res_strand$signatures$dbs) <- c("DBS A", "DBS B")
> colnames(nmf_res_strand$signatures$indel) <- c("INDEL A", "INDEL B")
```

Plot signatures with 192 features:

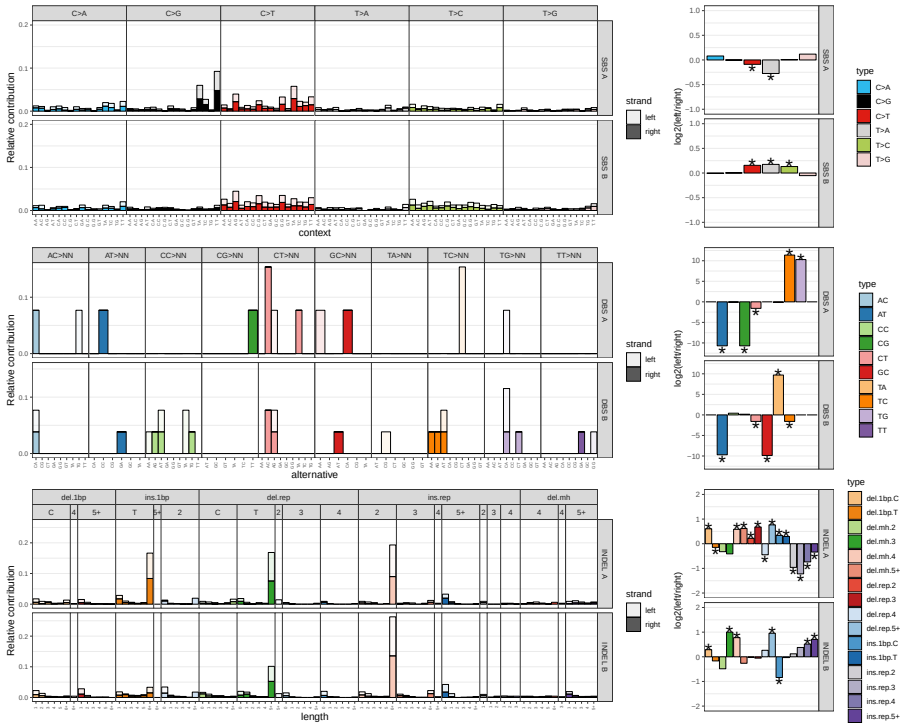
```
> a <- plot_strand_profiles(nmf_res_strand$signatures, condensed = TRUE,
+                           mode = "replication")
```

Plot strand bias per mutation type for each signature with significance test:

```
> b <- plot_signature_strand_bias(nmf_res_strand$signatures)
```

Combine the plots into one figure:

```
> grid.arrange(a, b, ncol = 2, widths = c(5, 1.8))
```



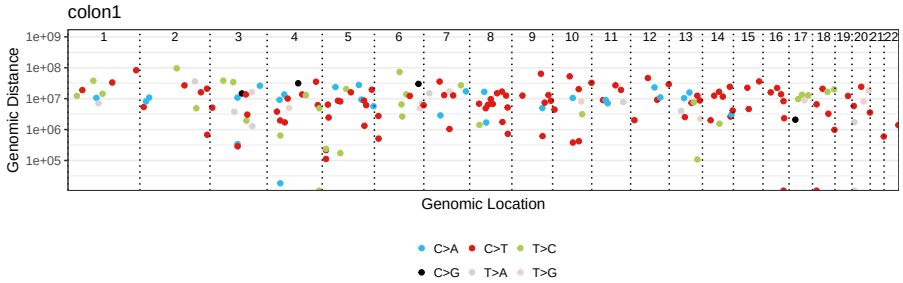
6 Genomic distribution

6.1 Rainfall plot

A rainfall plot visualizes mutation types and intermutation distance. Rainfall plots can be used to visualize the distribution of mutations along the genome or a subset of chromosomes. The y-axis corresponds to the distance of a mutation with the previous mutation and is log10 transformed. Drop-downs from the plots indicate clusters or "hotspots" of mutations.

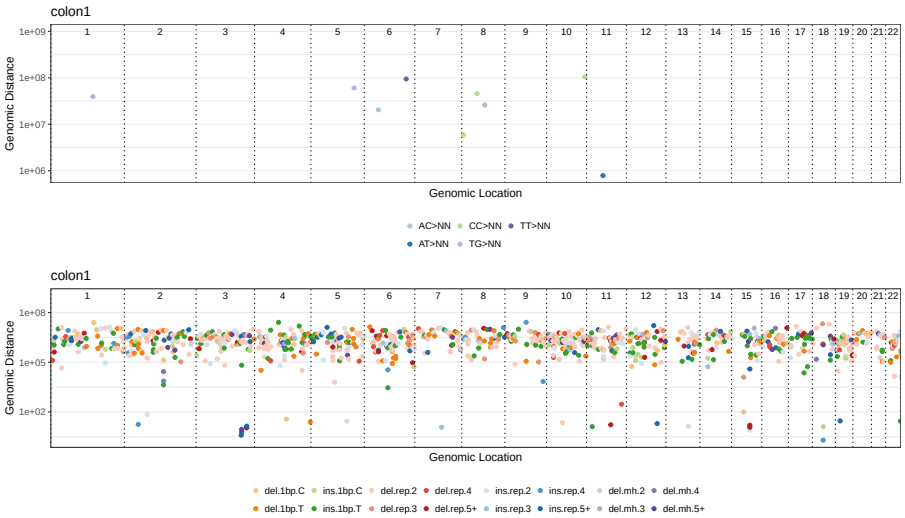
Make rainfall plot of single base substitutions from sample 1 over all autosomal chromosomes

```
> # Define autosomal chromosomes
> chromosomes <- seqnames(get(ref_genome))[1:22]
> # Make a rainfall plot
> plot_rainfall(vcfs[[1]], title = names(vcfs[1]),
+                 chromosomes = chromosomes, cex = 1.5, ylim = 1e+09)
```



Also make rainfall plots for DBS and indels:

```
> # Define autosomal chromosomes
> chromosomes <- seqnames(get(ref_genome))[1:22]
> # Make a rainfall plot
> plot_rainfall(vcfs_breast[[1]], title = names(vcfs[1]),
+               chromosomes = chromosomes,
+               type = c("dbs", "indel"),
+               cex = 1.5, ylim = 1e+09)
```



6.2 Enrichment or depletion of mutations in genomic regions

Test for enrichment or depletion of mutations in certain genomic regions, such as promoters, CTCF binding sites and transcription factor binding sites. To use your own genomic region definitions (based on e.g. ChipSeq experiments) specify your genomic regions in a named list of GRanges objects. Alternatively, use publicly available genomic annotation data, like in the example below.

6.2.1 Example: regulation annotation data from Ensembl using *biomaRt*

The following example displays how to download promoter, CTCF binding sites and transcription factor binding sites regions for genome build hg19 from Ensembl using *biomaRt*. For other datasets, see the *biomaRt* documentation (Durinck et al., 2005).

To install *biomaRt*, uncomment the following lines:

```
> source("https://bioconductor.org/biocLite.R")
> biocLite("biomaRt")
```

Load the *biomaRt* package.

```
> library(biomaRt)
```

Download genomic regions. NB: Here we take some shortcuts by loading the results from our example data. The corresponding code for downloading this data can be found above the command we run:

```
> # regulatory <- useEnsembl(biomart="regulation",
> #                          dataset="hsapiens_regulatory_feature",
> #                          GRCh = 37)
>
> ## Download the regulatory CTCF binding sites and convert them to
> ## a GRanges object.
> # CTCF <- getBM(attributes = c('chromosome_name',
> #                              'chromosome_start',
> #                              'chromosome_end',
> #                              'feature_type_name',
> #                              'cell_type_name'),
> #               filters = "regulatory_feature_type_name",
> #               values = "CTCF Binding Site",
> #               mart = regulatory)
> #
> # CTCF_g <- reduce(GRanges(CTCF$chromosome_name,
> #                 IRanges(CTCF$chromosome_start,
> #                 CTCF$chromosome_end)))
>
> CTCF_g <- readRDS(system.file("states/CTCF_g_data.rds",
+                   package="MutationalPatterns"))
> ## Download the promoter regions and convert them to a GRanges object.
>
> # promoter = getBM(attributes = c('chromosome_name', 'chromosome_start',
> #                                 'chromosome_end', 'feature_type_name'),
> #                  filters = "regulatory_feature_type_name",
> #                  values = "Promoter",
> #                  mart = regulatory)
> # promoter_g = reduce(GRanges(promoter$chromosome_name,
> #                     IRanges(promoter$chromosome_start,
> #                             promoter$chromosome_end)))
>
> promoter_g <- readRDS(system.file("states/promoter_g_data.rds",
+                       package="MutationalPatterns"))
```

```
> ## Download the promoter flanking regions and convert them to a GRanges object.
>
> # flanking = getBM(attributes = c('chromosome_name',
> #                                 'chromosome_start',
> #                                 'chromosome_end',
> #                                 'feature_type_name'),
> #                  filters = "regulatory_feature_type_name",
> #                  values = "Promoter Flanking Region",
> #                  mart = regulatory)
> # flanking_g = reduce(GRanges(
> #                         flanking$chromosome_name,
> #                         IRanges(flanking$chromosome_start,
> #                         flanking$chromosome_end)))
>
> flanking_g <- readRDS(system.file("states/promoter_flanking_g_data.rds",
+                                   package="MutationalPatterns"))
```

Combine all genomic regions (GRanges objects) in a named list:

```
> regions <- GRangesList(promoter_g, flanking_g, CTCF_g)
> names(regions) <- c("Promoter", "Promoter flanking", "CTCF")
```

Use the same chromosome naming convention consistently:

```
> seqlevelsStyle(regions) <- "UCSC"
```

6.3 Test for significant depletion or enrichment in genomic regions

It is necessary to include a list with Granges of regions that were surveyed in your analysis for each sample, that is: positions in the genome at which you have enough high quality reads to call a mutation. This can be determined using e.g. CallableLoci tool by GATK. If you would not include the surveyed area in your analysis, you might for example see a depletion of mutations in a certain genomic region that is solely a result from a low coverage in that region, and therefore does not represent an actual depletion of mutations.

We provided an example surveyed region data file with the package. For simplicity, here we use the same surveyed file for each sample. For a proper analysis, determine the surveyed area per sample and use these in your analysis.

Download the example surveyed region data:

```
> ## Get the filename with surveyed/callable regions
> surveyed_file <- system.file("extdata/callableloci-sample.bed",
+                              package = "MutationalPatterns")
> ## Import the file using rtracklayer and use the UCSC naming standard
> library(rtracklayer)
> surveyed <- import(surveyed_file)
> seqlevelsStyle(surveyed) <- "UCSC"
> ## For this example we use the same surveyed file for each sample.
> surveyed_list <- rep(list(surveyed), 9)
```

Test for enrichment or depletion of mutations in your defined genomic regions using a binomial test. For this test, the chance of observing a mutation is calculated as the total number of mutations, divided by the total number of surveyed bases.

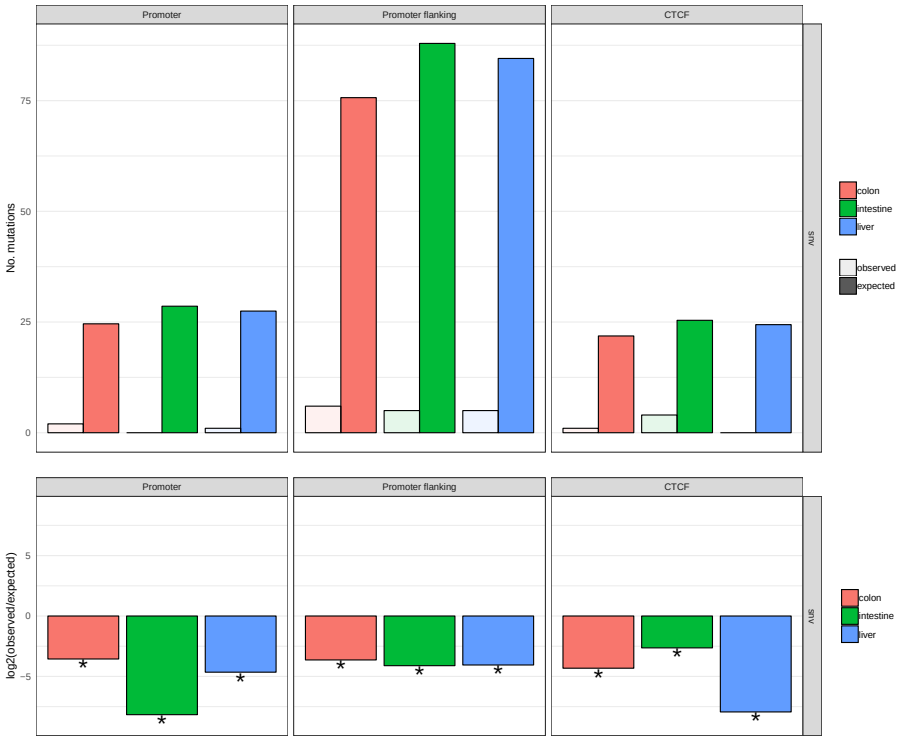
The vcf files including 3 types of cell tissue is used for SBS analyses:

```
> ## Calculate the number of observed and expected number of mutations in
> ## each genomic regions for each sample.
> distr <- genomic_distribution(vcfs_tissues, surveyed_list, regions)
```

```
> ## Perform the enrichment/depletion test by tissue type.
> distr_test <- enrichment_depletion_test(distr, by = tissue)
> head(distr_test)
```

```
         by            region mutation n_muts surveyed_length surveyed_region_length observed
1     colon          Promoter      snv   1248       727070334               14327310        2
2 intestine          Promoter      snv   1450       727070334               14327310        0
3     liver          Promoter      snv   1394       727070334               14327310        1
4     colon Promoter flanking      snv   1248       727070334               44087613        6
5 intestine Promoter flanking      snv   1450       727070334               44087613        5
6     liver Promoter flanking      snv   1394       727070334               44087613        5
          prob expected    effect         pval significant
1 1.716478e-06 24.59251 depletion 6.846365e-09           *
2 1.994305e-06 28.57303 depletion 3.898344e-13           *
3 1.917284e-06 27.46952 depletion 3.345879e-11           *
4 1.716478e-06 75.67540 depletion 3.857595e-25           *
5 1.994305e-06 87.92415 depletion 3.030213e-31           *
6 1.917284e-06 84.52846 depletion 7.442286e-30           *
```

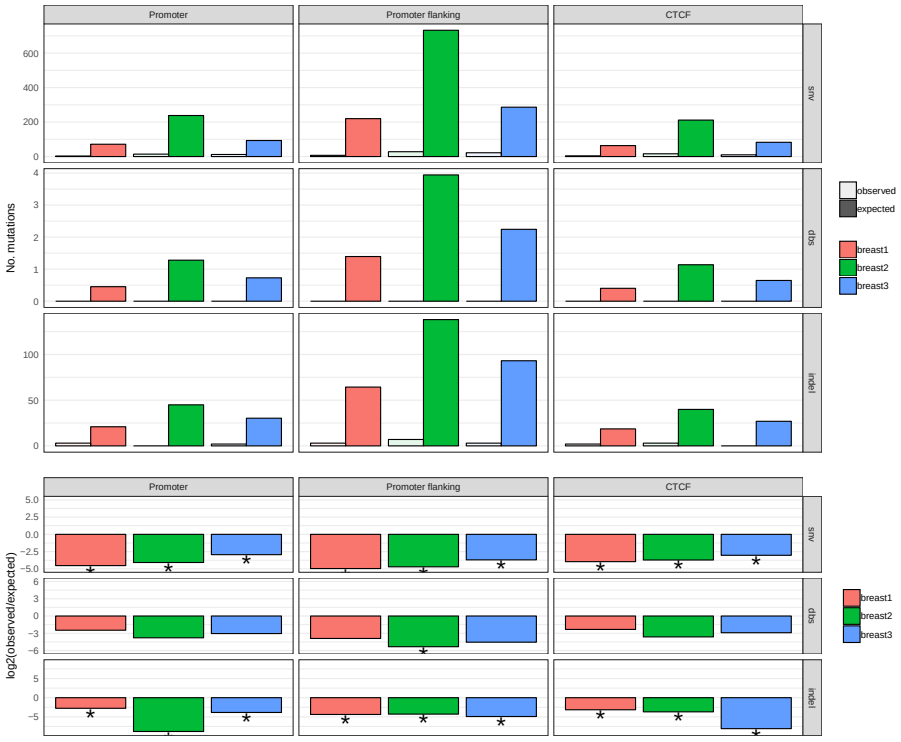
```
> plot_enrichment_depletion(distr_test)
```



The test can be repeated for DBS and indels. Use now the breast cancer organoid vcfs:

```
> ## For this example we use the same surveyed file for each sample.
> surveyed_list <- rep(list(surveyed), 3)
> distr <- genomic_distribution(vcfs_breast, surveyed_list, regions, type = "all")
> distr_test <- enrichment_depletion_test(distr)
> head(distr_test)

       by mutation   region n_muts surveyed_length surveyed_region_length observed         prob
1 breast1      snv Promoter   3627       242356778                4775770        3 1.496554e-05
2 breast1      dbs Promoter     23       242356778                4775770        0 9.490141e-08
3 breast1    indel Promoter   1061       242356778                4775770        3 4.377843e-06
4 breast2      snv Promoter  12083       242356778                4775770       14 4.985625e-05
5 breast2      dbs Promoter     65       242356778                4775770        0 2.681996e-07
6 breast2    indel Promoter   2279       242356778                4775770        0 9.403492e-06
     expected    effect         pval significant
1  71.4719759 depletion 5.787770e-27           *
2   0.4532273 depletion 6.355736e-01
3  20.9075728 depletion 1.466769e-06           *
4 238.1019808 depletion 8.949742e-82           *
5   1.2808598 depletion 2.777983e-01
6  44.9089145 depletion 3.134834e-20           *
```

```
> plot_enrichment_depletion(distr_test)
```



References

Blokzijl, F., de Ligt, J., Jager, M., Sasselli, V., Roerink, S., Sasaki, N., ... van Boxtel, R. (2016, Oct 13). Tissue-specific mutation accumulation in human adult stem cells during life. *Nature*, *538*(7624), 260–264. Retrieved from http://dx.doi.org/10.1038/nature19768 (Letter)

Durinck, S., Moreau, Y., Kasprzyk, A., Davis, S., De Moor, B., Brazma, A., & Huber, W. (2005, Aug 15). Biomart and bioconductor: a powerful link between biological databases and microarray data analysis. *Bioinformatics*, *21*(16), 3439–3440. Retrieved from http://dx.doi.org/10.1093/bioinformatics/bti525 doi: 10.1093/bioinformatics/bti525

Gaujoux, R., & Seoighe, C. (2010). A flexible r package for nonnegative matrix factorization. *BMC Bioinformatics*, *11*(1), 367. Retrieved from http://dx.doi.org/10.1186/1471-2105-11-367 doi: 10.1186/1471-2105-11-367

Rosenthal, R., McGranahan, N., Herrero, J., Taylor, B. S., & Swanton, C. (2016, February). deconstructSigs: delineating mutational processes in single tumors distinguishes DNA repair deficiencies and patterns of carcinoma evolution. *Genome Biology*, *17*(1). Retrieved from https://doi.org/10.1186/s13059-016-0893-4 doi: 10.1186/s13059-016-0893-4

Sachs, N., de Ligt, J., Kopper, O., Gogola, E., Bounova, G., Weeber, F., ... others (2018). A living biobank of breast cancer organoids captures disease heterogeneity. *Cell*, *172*(1-2), 373–386.

7 Session Information

- R version 3.4.3 (2017-11-30), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=en_US.UTF-8`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=nl_NL.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 16.04.6 LTS`
- Matrix products: default
- BLAS: `/home/cog/bvanderroest/R/R-3.4.3/lib/libRblas.so`
- LAPACK: `/home/cog/bvanderroest/R/R-3.4.3/lib/libRlapack.so`
- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.40.0, Biobase 2.38.0, BiocGenerics 0.24.0, biomaRt 2.34.2, Biostrings 2.46.0, BSgenome 1.46.0, BSgenome.Hsapiens.UCSC.hg19 1.4.0, cluster 2.0.7-1, doParallel 1.0.14, foreach 1.4.4, GenomeInfoDb 1.14.0, GenomicFeatures 1.30.3, GenomicRanges 1.30.3, ggplot2 3.1.0, gridExtra 2.3, IRanges 2.12.0, iterators 1.0.10, MutationalPatterns 1.6.2, NMF 0.21.0, pkgmaker 0.27, registry 0.5, rngtools 1.3.1, rtracklayer 1.38.3, S4Vectors 0.16.0, testthat 2.0.1, TxDb.Hsapiens.UCSC.hg19.knownGene 3.2.2, XVector 0.18.0
- Loaded via a namespace (and not attached): assertthat 0.2.0, backports 1.1.3, bibtex 0.4.2, bindr 0.1.1, bindrcpp 0.2.2, BiocInstaller 1.28.0, BiocParallel 1.12.0, BiocStyle 2.6.1, bit 1.1-14, bit64 0.9-7, bitops 1.0-6, blob 1.1.1, callr 3.3.2, cli 1.0.1, codetools 0.2-16, colorspace 1.4-0, compiler 3.4.3, cowplot 0.9.4, crayon 1.3.4, DBI 1.0.0, deconstructSigs 1.8.0, DelayedArray 0.4.1, desc 1.2.0, devtools 2.2.1.9000, digest 0.6.18, dplyr 0.7.8, ellipsis 0.3.0, evaluate 0.14, fs 1.3.1, GenomeInfoDbData 1.0.0, GenomicAlignments 1.14.2, ggdendro 0.1-20, glue 1.3.0, grid 3.4.3, gridBase 0.4-7, gtable 0.2.0, hms 0.4.2, htmltools 0.3.6, httr 1.4.0, knitr 1.25, labeling 0.3, lattice 0.20-38, lazyeval 0.2.1, magrittr 1.5, MASS 7.3-51.1, Matrix 1.2-15, matrixStats 0.54.0, memoise 1.1.0, munsell 0.5.0, pillar 1.3.1, pkgbuild 1.0.6, pkgconfig 2.0.2, pkgload 1.0.2, plyr 1.8.4, pracma 2.2.2, prettyunits 1.0.2, processx 3.4.1, progress 1.2.0, ps 1.3.0, purrr 0.2.5, R6 2.3.0, RColorBrewer 1.1-2, Rcpp 1.0.0, RCurl 1.95-4.11, remotes 2.1.0, reshape2 1.4.3, rlang 0.4.0, rmarkdown 1.16, RMySQL 0.10.16, rprojroot 1.3-2, Rsamtools 1.30.0, RSQLite 2.1.1, rstudioapi 0.9.0, scales 1.0.0, sessioninfo 1.1.1, stringi 1.2.4, stringr 1.3.1, SummarizedExperiment 1.8.1, tibble 2.0.1, tidyselect 0.2.5, tools 3.4.3, usethis 1.5.1, VariantAnnotation 1.24.5, withr 2.1.2, xfun 0.10, XML 3.98-1.16, xtable 1.8-3, yaml 2.2.0, zlibbioc 1.24.0