Introduction to *MutationalPatterns*

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

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

October 17, 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).

Load the MutationalPatterns package:

```
> library(MutationalPatterns)
```

Locate the VCF files of the example data:

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

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)
> 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] "C>A" "C>A" "G>A" "G>C" "G>T" "C>A" "G>T" "G>C" "A>G" "G>T" "C>T" "C>T"
```

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] "C>A" "C>A" "C>T" "C>G" "C>A" "C>A" "C>A" "C>G" "T>C" "C>A" "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  chr1  chr1  chr1
"TCT" "CCA" "TGG" "CGC" "GGA" "CCC" "CGT" "TGT" "CAG" "GGA" "CCT" "GCA"
```

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] "C>A" "C>A" "C>T" "C>G" "C>A" "C>A" "C>A" "C>G" "T>C" "C>A" "C>T" "C>T"

$context
 chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1
"TCT" "CCA" "CCA" "GCG" "TCC" "CCC" "ACG" "ACA" "CTG" "TCC" "CCT" "GCA"
```

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	19088	3604	5477	4729	3356	975	371	5106
colon2	1319	1186	2341	977	2033	749	200	2141
colon3	1581	2184	7282	1200	4162	1160	854	6428
intestine1	28599	5711	10136	7190	6636	2007	740	9396
intestine2	899	540	30322	1402	2237	819	2681	27641
intestine3	601	884	2331	465	1057	367	283	2048
liver1	1388	889	2663	1136	2180	678	350	2313
liver2	1737	1139	2918	1339	2522	937	295	2623
liver3	5064	1413	29800	5605	15826	3560	2201	27599

3.2 Double base substitutions and indels

Not only single base substitutions can be retrieved from the VCF GRanges object, but there is also the option to extract double base substitutions and/or indels, if they are present in the loaded VCF files. When multiple mutation types are requested, the output will be a list of mutation types.

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:

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

$dbs
 [1] "CC>TA" "GG>TT" "GG>TT" "CC>AA" "GG>CT" "CG>AT" "GG>TT" "GC>TA" "CC>AA" "CC>AA" "CC>AG"
[12] "CC>AA"

$indel
 [1] "AT>A"    "T>TA"    "TGATA>T" "TG>T"    "A>ATAT"  "TAC>T"   "TG>T"    "AAT>A"   "AG>A"
[10] "TG>T"    "GGA>G"   "C>CAT"
```

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

$dbs
 [1] "CC>TA" "CC>AA" "CC>AA" "CC>AA" "CC>AG" "CG>AT" "CC>AA" "GC>TA" "CC>AA" "CC>AA" "CC>AG"
[12] "CC>AA"

$indel
 [1] "AT>A"    "T>TA"    "TGATA>T" "TG>T"    "A>ATAT"  "TAC>T"   "TG>T"    "AAT>A"   "AG>A"
[10] "TG>T"    "GGA>G"   "C>CAT"
```

The insertions and deletions can be translated 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, use:

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

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

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

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

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[[1]], ref_genome, type = c("dbs","indel"))
> lapply(type_context, function(x) lapply(x, head, 12))

$dbs
$dbs$types
 [1] "CC>TA" "CC>AA" "CC>AA" "CC>AA" "CC>AG" "CG>AT" "CC>AA" "GC>TA" "CC>AA" "CC>AA" "CC>AG"
[12] "CC>AA"


$indel
$indel$types
 [1] "AT>A"    "T>TA"    "TGATA>T" "TG>T"    "A>ATAT"  "TAC>T"   "TG>T"    "AAT>A"   "AG>A"
[10] "TG>T"    "GGA>G"   "C>CAT"

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

3.3 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:

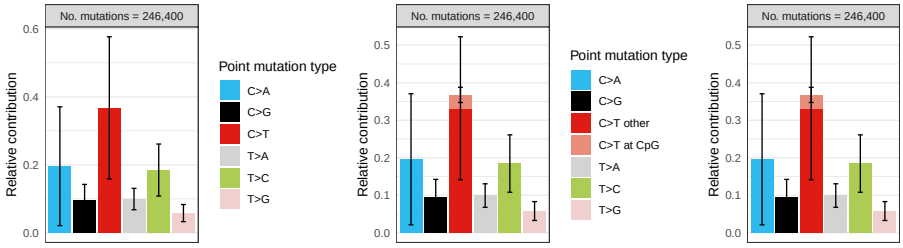
```
> p2 <- plot_spectrum(type_occurrences, CT = TRUE)
```

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))
```



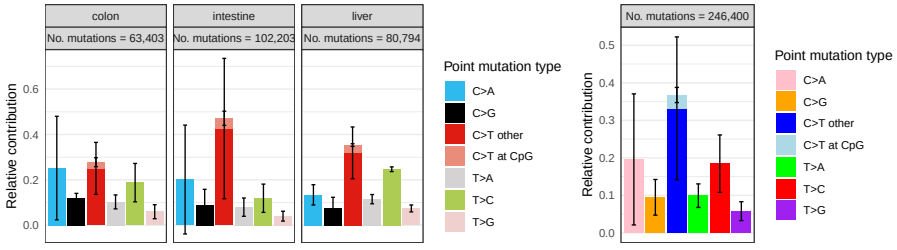
You can facet the per sample group, e.g. plot the spectrum for each tissue separately:

```
> 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))
```



3.4 96 mutational profile

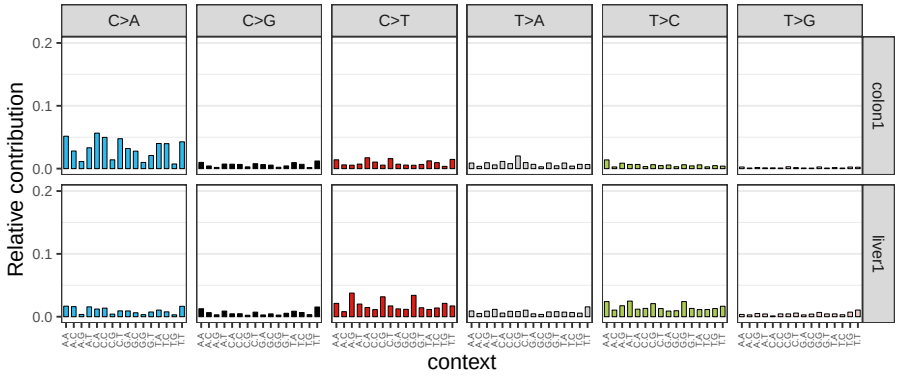
Make a 96 trinucleodide mutation count matrix:

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

        colon1 colon2 colon3 intestine1 intestine2 intestine3 liver1 liver2 liver3
A[C>A]A   1922    146    156       2706         98         65    150    202    285
```

```
A[C>A]C   1044     76    141       1605        40         51    143    113    252
A[C>A]G    421     24     60        731         7         13     32     31     28
A[C>A]T   1235     94     84       1716        42         36    140    122    187
C[C>A]A   2105     86     98       3202       145         38    109    150    337
C[C>A]C   1852     90     64       2900        66         39    122    123    821
```

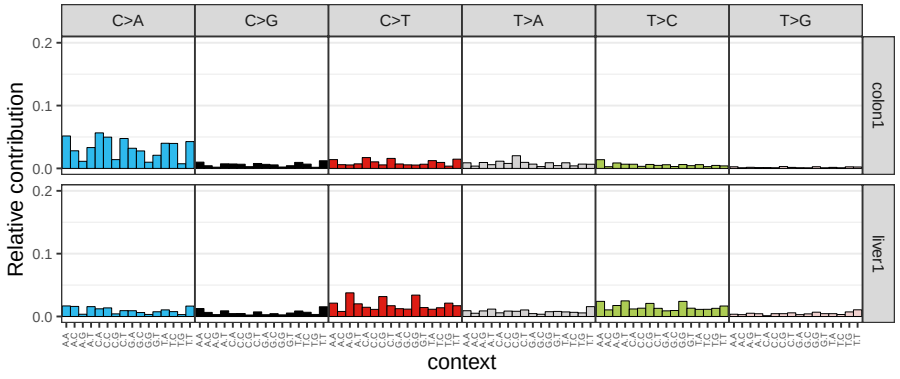
Plot the 96 profile of two samples:

```
> plot_profiles(mut_mat[,c(1,7)])
```



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

```
> plot_profiles(mut_mat[,c(1,7)], condensed = TRUE)
```



3.5 Plot mutation profiles of different types

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, ref_genome = ref_genome, type = "all")
> lapply(mut_mat, head)
```

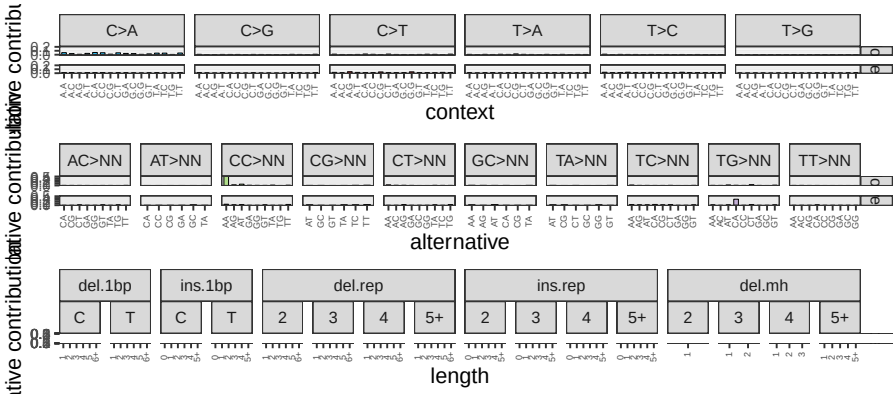
```
$snv
        colon1 colon2 colon3 intestine1 intestine2 intestine3 liver1 liver2 liver3
A[C>A]A   1922    146    156       2706         98         65    150    202    285
A[C>A]C   1044     76    141       1605         40         51    143    113    252
A[C>A]G    421     24     60        731          7         13     32     31     28
A[C>A]T   1235     94     84       1716         42         36    140    122    187
C[C>A]A   2105     86     98       3202        145         38    109    150    337
C[C>A]C   1852     90     64       2900         66         39    122    123    821

$dbs
      colon1 colon2 colon3 intestine1 intestine2 intestine3 liver1 liver2 liver3
AC>CA      3      0      2          0          0          0      0      1     35
AC>CG      2      0      0          0          0          0      0      0      0
AC>CT      2      0      0          3          1          0      1      0      1
AC>GA      1      0      0          1          2          0      0      1      2
AC>GG      0      0      0          0          1          0      0      1      0
AC>GT      1      1      1          1          3          1      1      1      7

$indel
                         colon1 colon2 colon3 intestine1 intestine2 intestine3 liver1 liver2
del.1bp.homopol.C.len.1     145     29     13        235         22         14     35     32
del.1bp.homopol.C.len.2     181     10      9        263         11         13     17     25
del.1bp.homopol.C.len.3     147      5      2        204          3          8      6     13
del.1bp.homopol.C.len.4      57      2      4         70          3          5      2      1
del.1bp.homopol.C.len.5      39      5      1         52          1          0      1      6
del.1bp.homopol.C.len.6+     21     17      2         43          5         14     10     14
                         liver3
del.1bp.homopol.C.len.1     339
del.1bp.homopol.C.len.2     113
del.1bp.homopol.C.len.3      90
del.1bp.homopol.C.len.4     251
del.1bp.homopol.C.len.5     723
del.1bp.homopol.C.len.6+   2870
```

Make a list of two samples:

```
> mut_mat_sub <- list("snv" = mut_mat$snv[,c(1,7)],
+                     "dbs" = mut_mat$dbs[,c(1,7)],
+                     "indel" = mut_mat$indel[,c(1,7)])
```

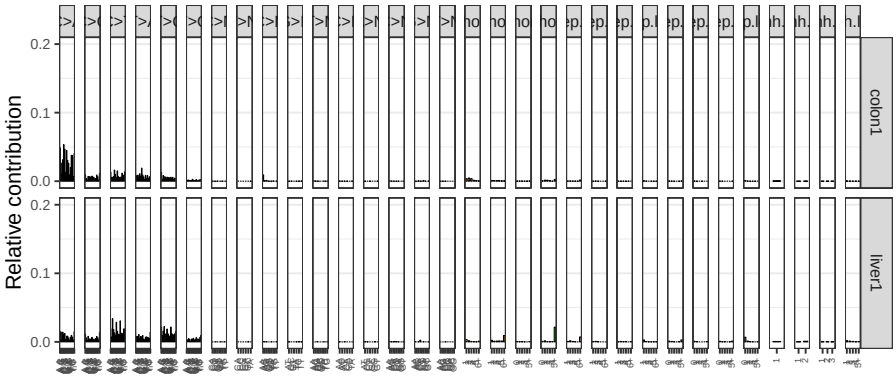
Plot the mutation profiles of the two samples:

```
> plot_profiles(mut_mat_sub, type = "all")
```



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

```
> plot_profiles(mut_mat_sub, type = "all", 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:

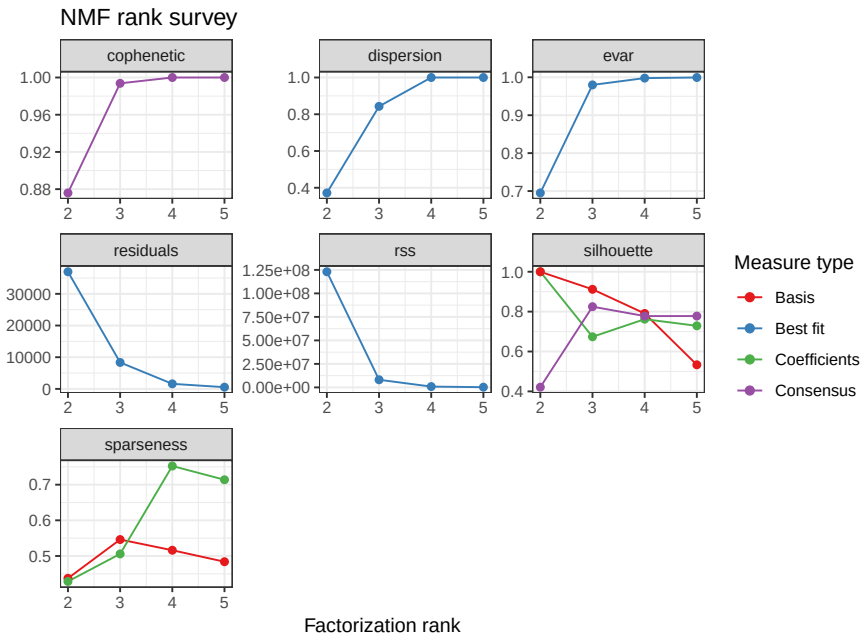
```
> mut_mat <- mut_matrix(vcf_list = vcfs, ref_genome = ref_genome)
> mut_mat <- mut_mat + 0.0001
```

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):

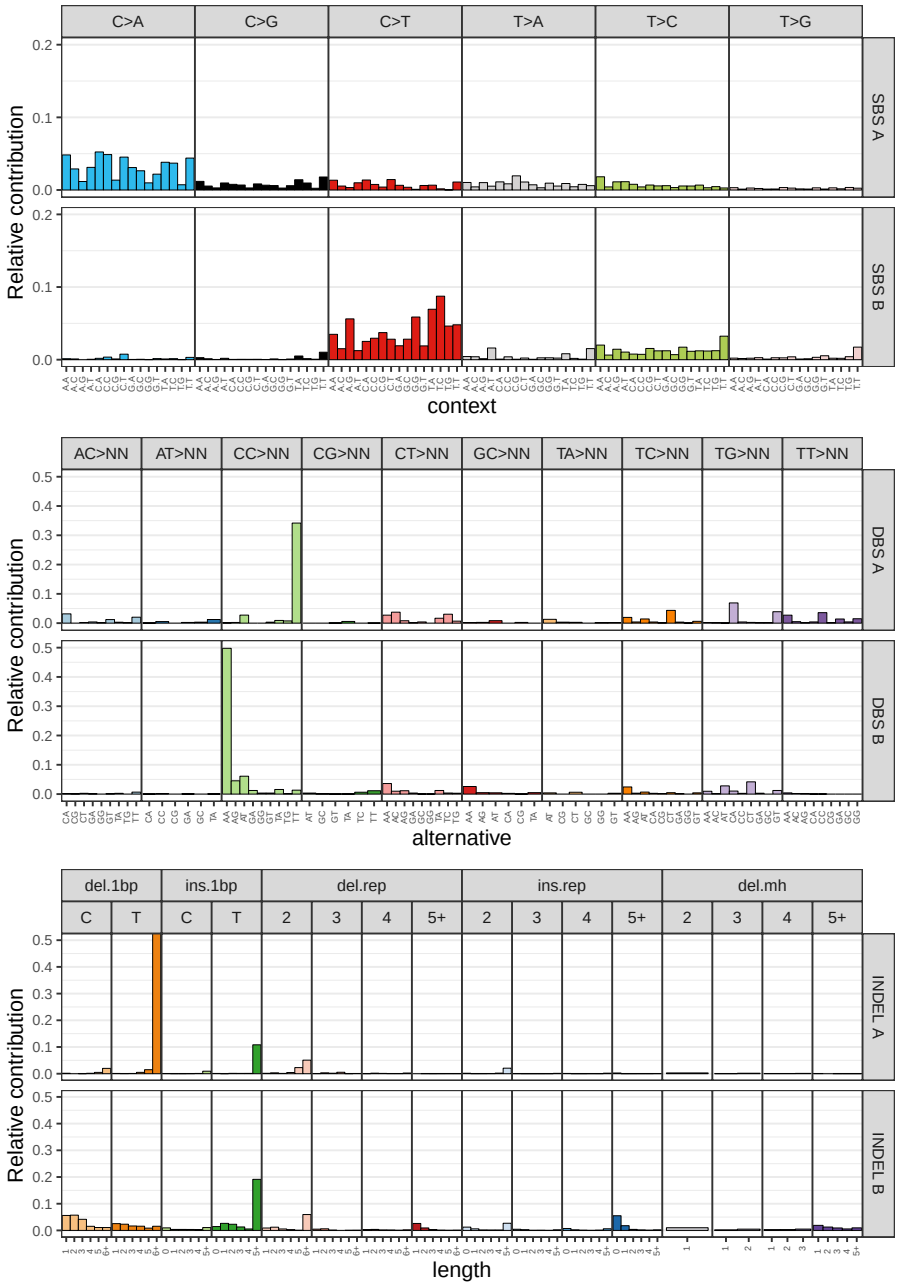
```
> nmf_res <- extract_signatures(mut_mat, rank = 2, nrun = 10)
```

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 DBS and indels, make a list of mutation matrices for each mutation type:

```
> mut_mat <- mut_matrix(vcf_list = vcfs, 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`.

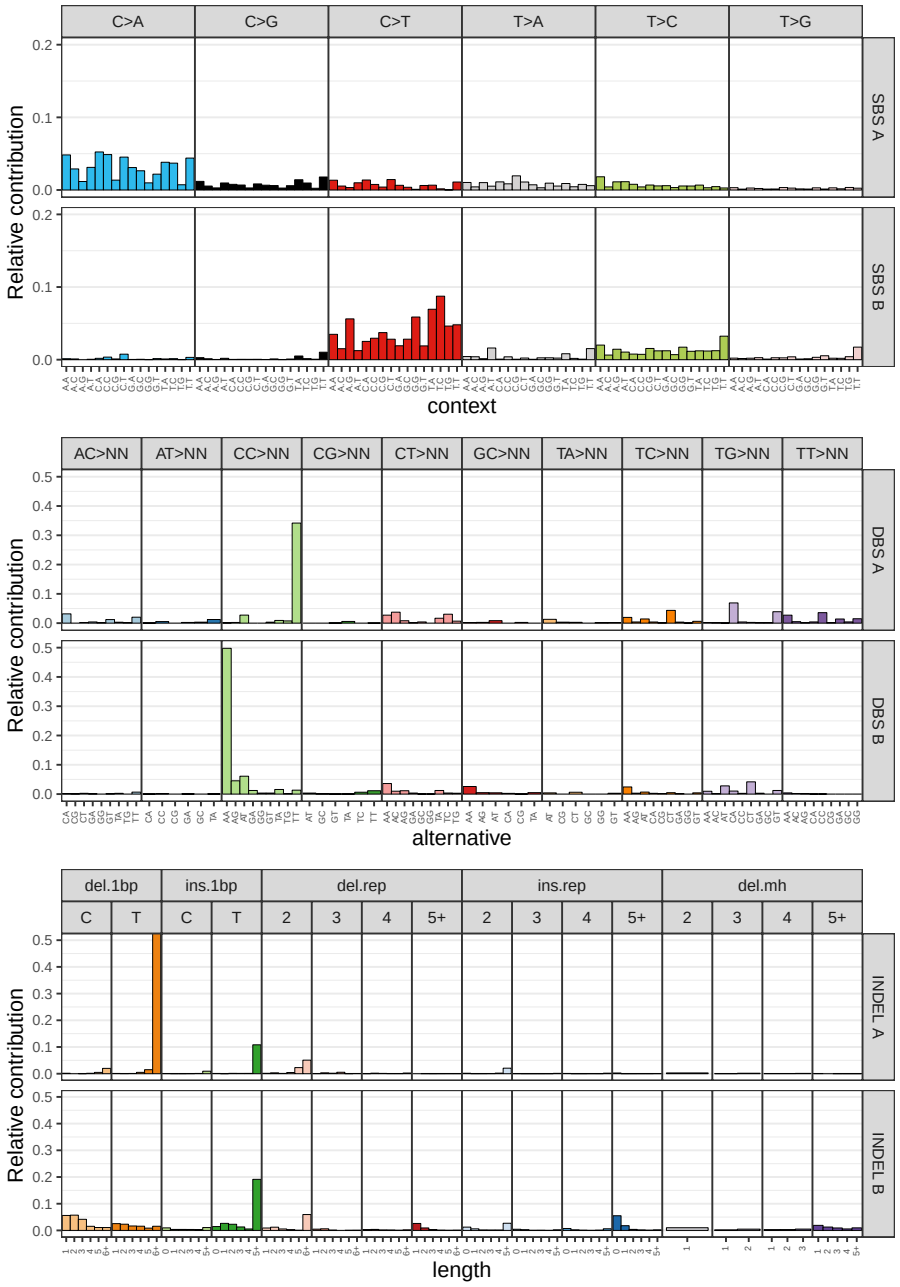
```
> nmf_res <- extract_signatures(mut_mat,
+                               rank = c("snv" = 2, "dbs" = 2, "indel" = 2),
+                               type = "all",
+                               nrun = 10)
```

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, type = "all")
```



Visualize the contribution of the SBS signatures in a barplot:

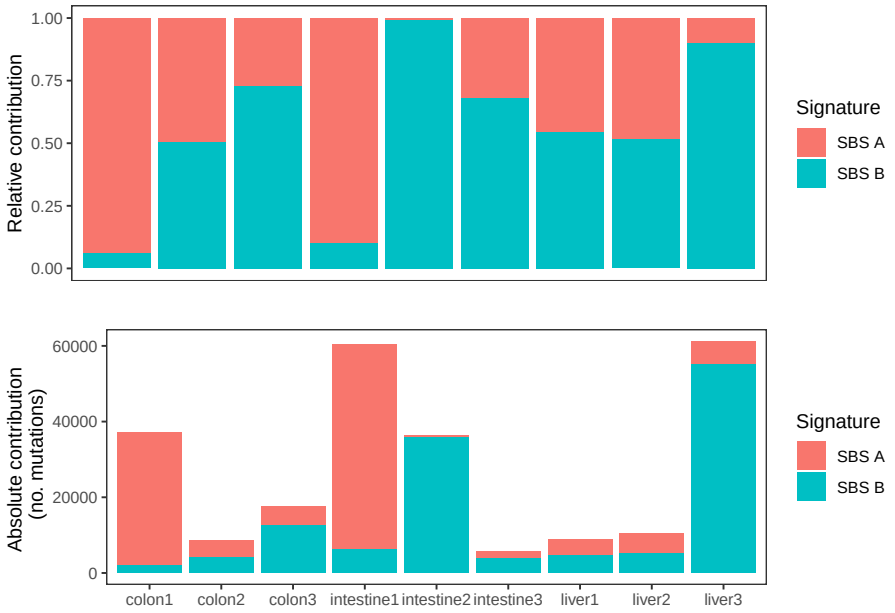
```
> pc1 <- plot_contribution(nmf_res$contribution, nmf_res$signature,
+                          mode = "relative")
```

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

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

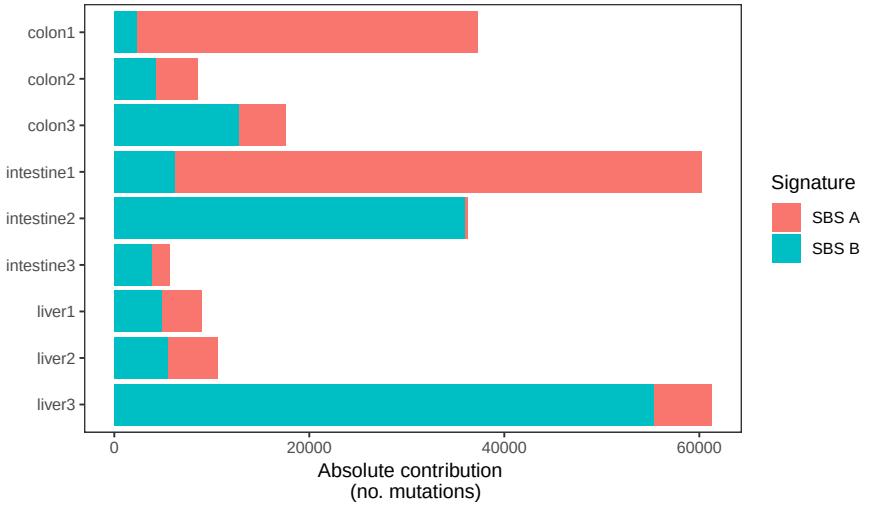
Combine the two plots:

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



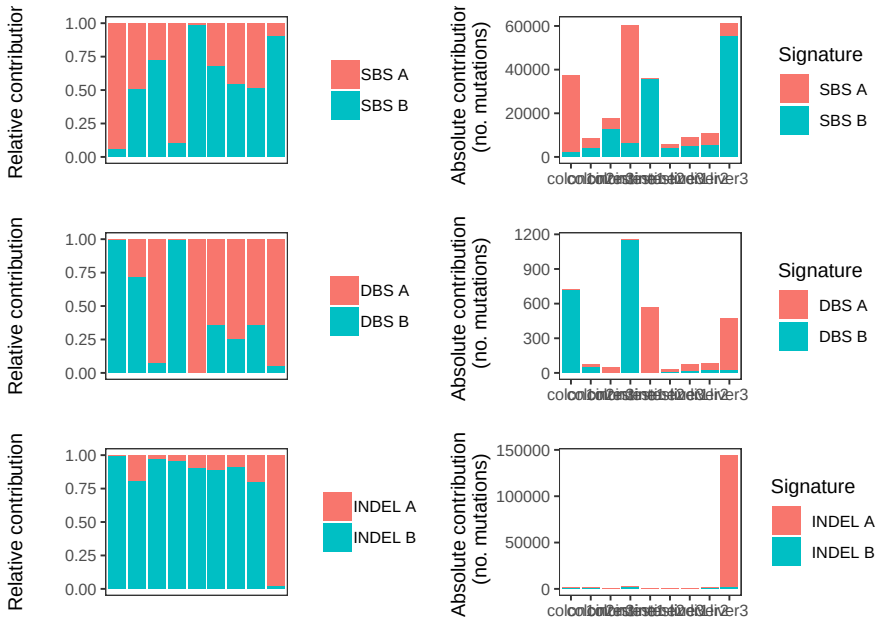
Flip X and Y coordinates:

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



Visualize the contribution of all signatures in both relative and absolute number of mutations:

```
> 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:

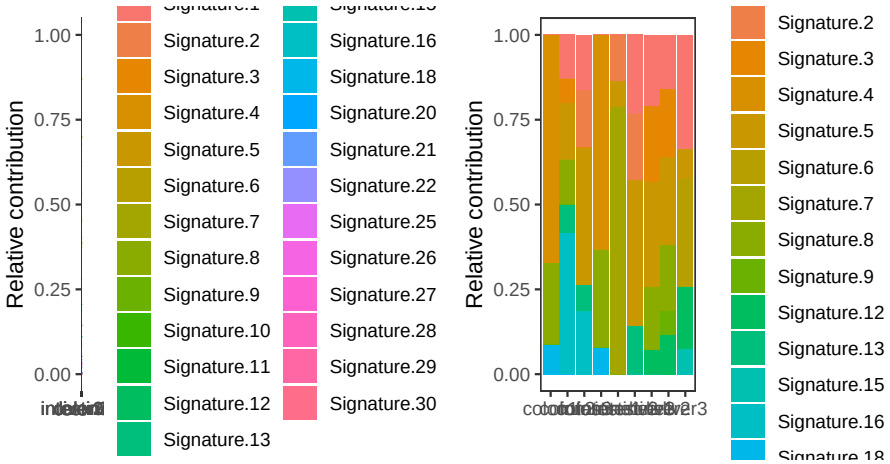
```
> pch1 <- plot_contribution_heatmap(nmf_res$contribution,
+                                   sig_order = c("SBS B", "SBS A"))
```

Plot SBS signature contribution as a heatmap without sample clustering:

```
> pch2 <- plot_contribution_heatmap(nmf_res$contribution, cluster_samples=FALSE)
```

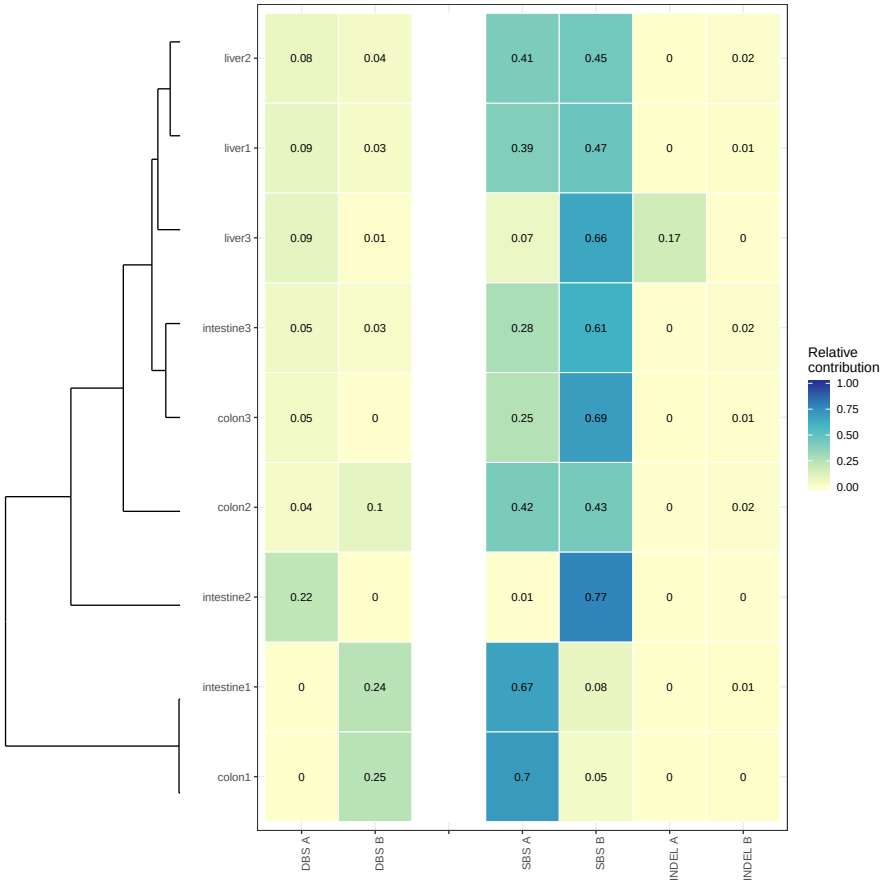
Combine the plots into one figure:

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



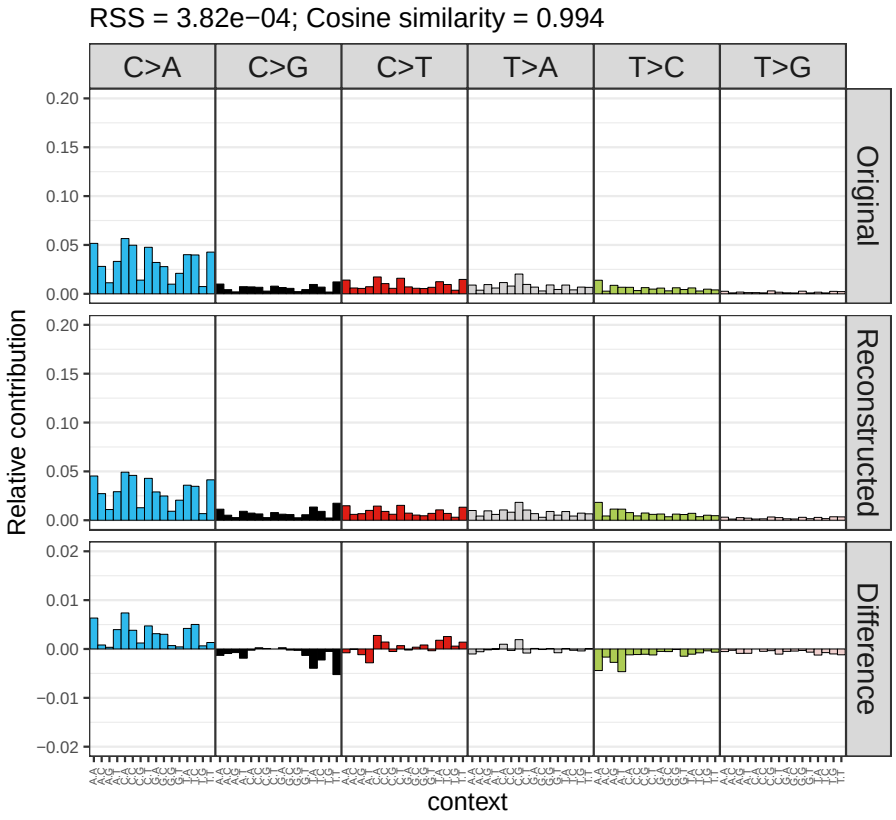
When plotting the signature contribution of multiple mutation types, it is possible to cluster on a specified mutation type. The mutation type(s) on which the data will be clustered, will show up at the left side of the heatmap.

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



Compare the reconstructed mutational profile with the original mutational profile:

```
> plot_compare_profiles(mut_mat$snv[,1],
+                       nmf_res$snv$reconstructed[,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. For convenience the SBS signatures of version 2 are used:

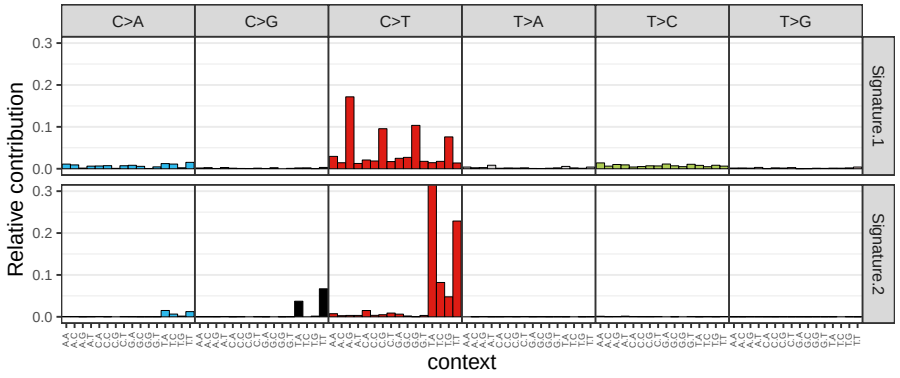
```
> sp_url <- paste("https://cancer.sanger.ac.uk/cancergenome/assets/",
+                 "signatures_probabilities.txt", sep = "")
> snv_signatures = read.table(sp_url, sep = "\t", header = TRUE)
> # Match the order of the mutation types to MutationalPatterns standard
> new_order = match(row.names(mut_mat$snv), snv_signatures$Somatic.Mutation.Type)
> # 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$Somatic.Mutation.Type
> # Keep only 96 contributions of the signatures in matrix
> snv_signatures = as.matrix(snv_signatures[,4:33])
```

DBS and indel signatures are available from synapse.org ID syn12009743. Download the reference whole-genome signatures.

```
> dbs_signatures = read.csv("sigProfiler_DBS_signatures.csv")
> rownames(dbs_signatures) = dbs_signatures$Mutation.Type
> dbs_signatures = as.matrix(dbs_signatures[,2:11])
> indel_signatures = read.csv("sigProfiler_ID_signatures.csv")
> rownames(indel_signatures) = INDEL_COSMIC
> indel_signatures = as.matrix(indel_signatures[,2:18])
> cancer_signatures = list("snv" = snv_signatures,
+                          "dbs" = dbs_signatures,
+                          "indel" = indel_signatures)
```

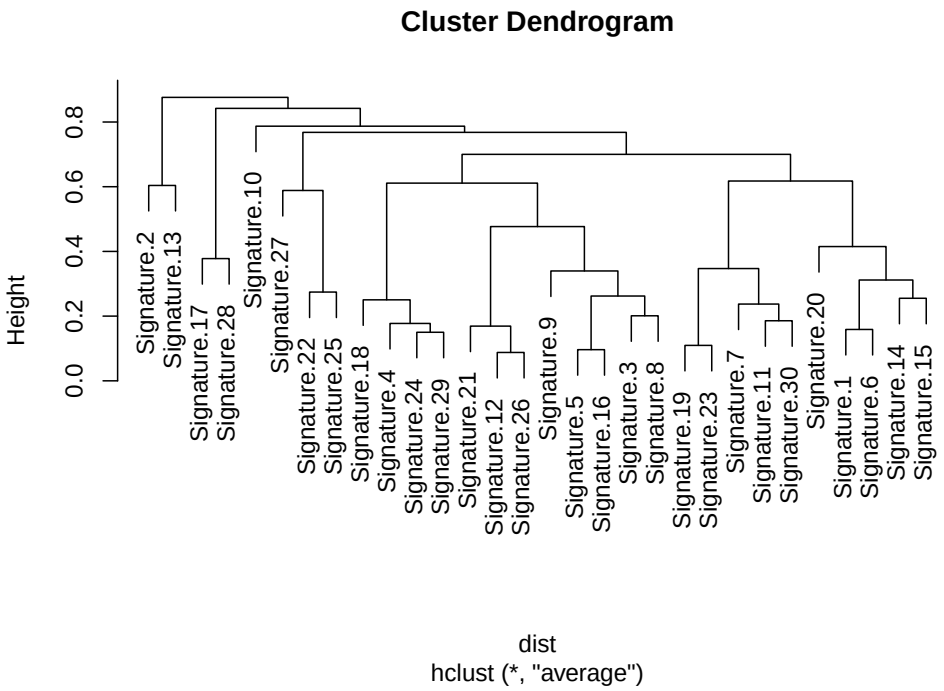
Plot mutational profile of the first two COSMIC SBS signatures:

```
> plot_profiles(cancer_signatures$snv[,1:2], condensed = TRUE, ymax = 0.3)
```



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

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



4.2.2 Similarity between mutational profiles and COSMIC signatures

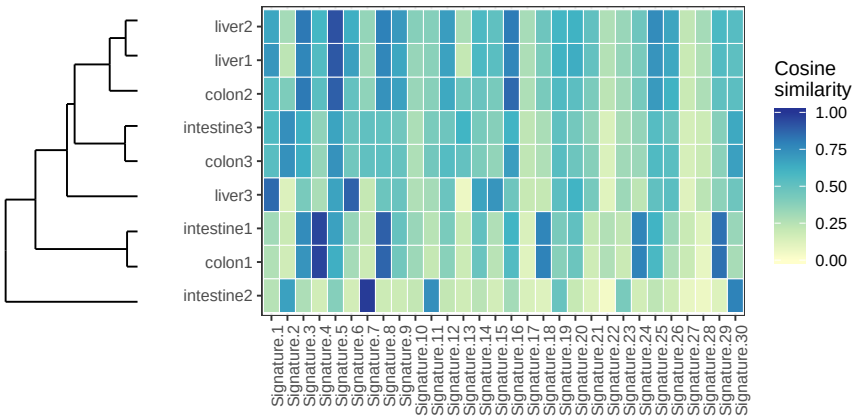
The similarity between each mutational profile and each COSMIC signature, can be calculated with `cos_sim_matrix`, and visualized with `plot_cosine_heatmap`. The cosine similarity reflects how well each mutational profile can be explained by each signature individually. The advantage of this heatmap 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$snv[,1], cancer_signatures$snv[,1])

[1] 0.2643801
```

Calculate pairwise cosine similarity between mutational profiles of single base substitutions and COSMIC signatures:

```
> cos_sim_samples_signatures = cos_sim_matrix(mut_mat, cancer_signatures,
+                                             type = "all")
> # Plot heatmap with specified signature order
> 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.

Fit mutation matrix to the COSMIC SBS mutational signatures:

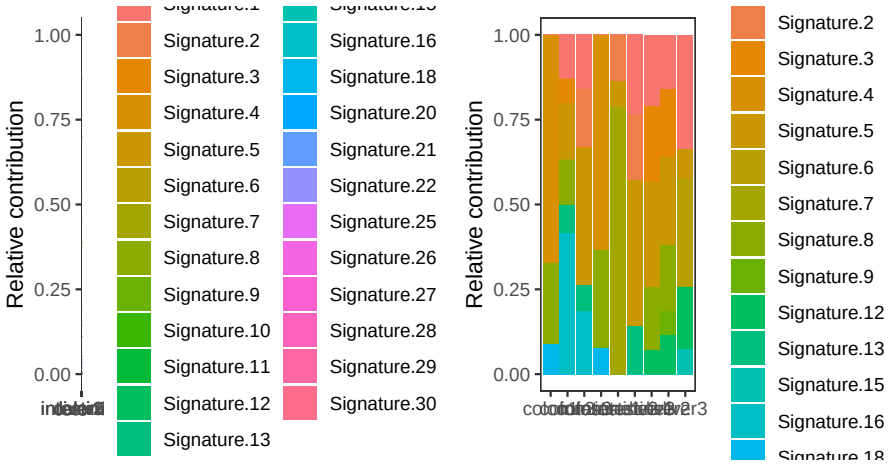
```
> fit_res <- fit_to_signatures(mut_mat, cancer_signatures)
```

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

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

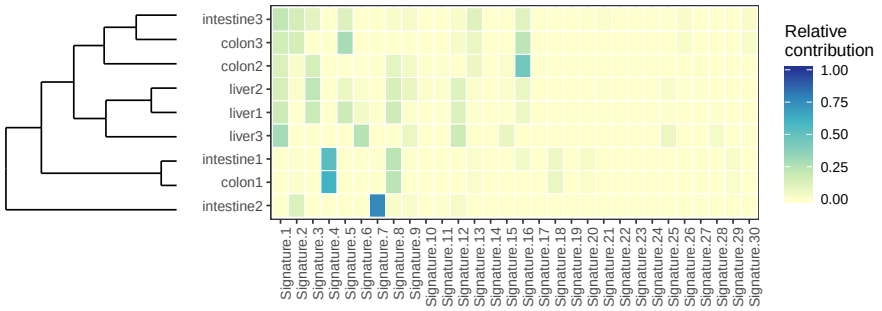
Results of the golden ratio search algorithm are only relative, so fit the mutation matrix with the golden ratio search and plot results from both methods in relative contribution:

```
> fit_res_grs <- fit_to_signatures(mut_mat, cancer_signatures,
+                                  method = "golden-ratio-search")
> # Select signatures with some contribution
> select_grs <- which(rowSums(fit_res_grs$contribution) > 0.06)
> # Make a color vector for SBS signatures such that colors will match in
> # results from both algorithms
> colorvector <- default_colors_ggplot(ncol(cancer_signatures$snv))
> # Plot relative contribution from non-negative least squares
> pc1 <- plot_contribution(fit_res$contribution[select,],
+                          cancer_signatures$snv[,select],
+                          coord_flip = FALSE,
+                          mode = "relative",
+                          palette = list("snv" = colorvector[select]))
> # Plot relative contribution from golden ratio search
> pc2 <- plot_contribution(fit_res_grs$contribution[select_grs,],
+                                cancer_signatures$snv[,select_grs],
+                                coord_flip = FALSE,
+                                mode = "relative",
+                                palette = list("snv" = colorvector[select_grs]))
```



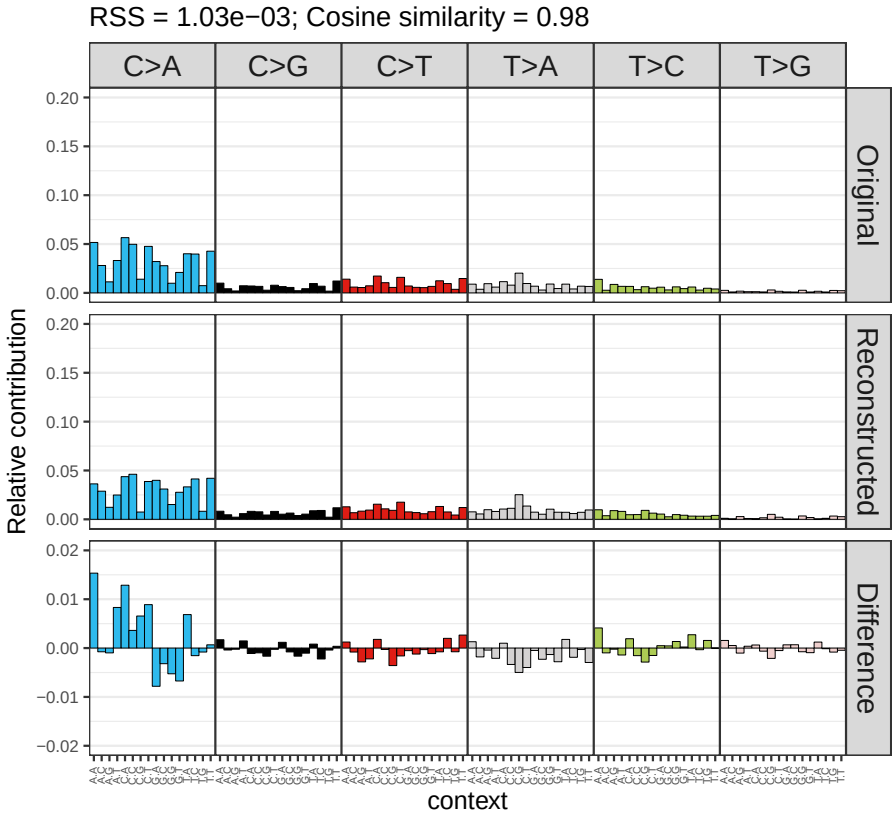
Plot relative contribution fitted with the non-negative least-squares problem of the SBS cancer signatures in each sample as a heatmap with sample clustering:

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



Compare the reconstructed mutational profile of sample 1 with its original mutational profile:

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



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

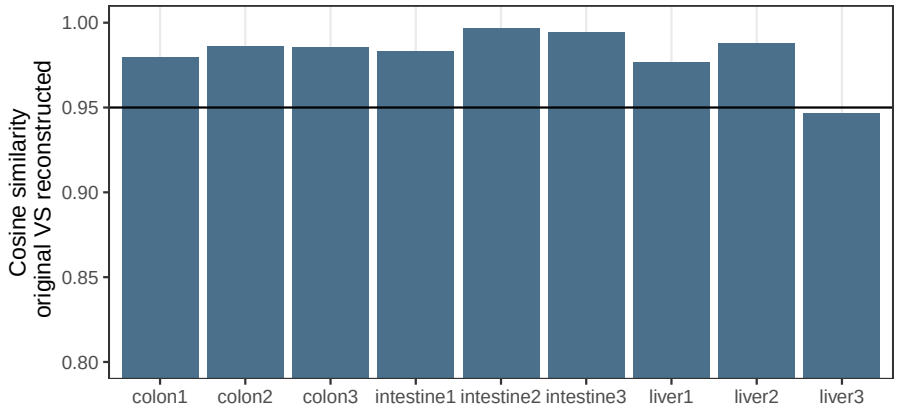
```
> # calculate all pairwise cosine similarities
> cos_sim_ori_rec <- cos_sim_matrix(mut_mat$snv, fit_res$reconstructed)
> # extract cosine similarities per sample between original and reconstructed
```

```
> cos_sim_ori_rec <- as.data.frame(diag(cos_sim_ori_rec))
```

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
> colnames(cos_sim_ori_rec) = "cos_sim"
> cos_sim_ori_rec$sample = row.names(cos_sim_ori_rec)
```

```
> # Load ggplot2
> library(ggplot2)
> # Make barplot
> ggplot(cos_sim_ori_rec, 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.

```
> strand = mut_strand(vcfs[[1]], genes_hg19, type = c("snv", "dbs"))
> lapply(strand, head, 10)

$snv
 [1] transcribed transcribed -           -           -           transcribed -
 [8] -           -           -
```

```
Levels: untranscribed transcribed -

$dbs
 [1] -             -             -             -             transcribed   -
 [7] 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, ref_genome, genes_hg19,
+                                  type = c("snv", "dbs"))
> lapply(mut_mat_s, function(x) x[1:5,1:5])

$snv
                      colon1 colon2 colon3 intestine1 intestine2
A[C>A]A-untranscribed    195     17     21        253          6
A[C>A]A-transcribed      285     45     30        496         19
A[C>A]C-untranscribed    111      8     26        179          6
A[C>A]C-transcribed      188     25     27        298          9
A[C>A]G-untranscribed     47      2     10         93          1

$dbs
                    colon1 colon2 colon3 intestine1 intestine2
AC>CA-untranscribed      0      0      1          0          0
AC>CA-transcribed        0      0      1          0          0
AC>CG-untranscribed      0      0      0          0          0
AC>CG-transcribed        0      0      0          0          0
AC>CT-untranscribed      0      0      0          0          1
```

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

```
> strand_counts <- strand_occurrences(mut_mat_s, by=tissue,
+                                     type = c("snv", "dbs"))
> lapply(strand_counts, head)

$snv
   group mutation type        strand no_mutations relative_contribution
1  colon      snv  C>A   transcribed         3893            0.18214570
4  colon      snv  C>A untranscribed         2543            0.11898189
7  colon      snv  C>G   transcribed         1245            0.05825106
10 colon      snv  C>G untranscribed         1174            0.05492912
13 colon      snv  C>T   transcribed         2813            0.13161465
16 colon      snv  C>T untranscribed         2847            0.13320545

$dbs
   group mutation type        strand no_mutations relative_contribution
1  colon      dbs   AC   transcribed            4           0.016194332
4  colon      dbs   AC untranscribed            3           0.012145749
7  colon      dbs   AT   transcribed            1           0.004048583
10 colon      dbs   AT untranscribed            1           0.004048583
13 colon      dbs   CC   transcribed           94           0.380566802
```

```
16 colon      dbs   CC untranscribed           49           0.198380567
```

Perform Poisson test for strand asymmetry significance testing:

```
> strand_bias <- strand_bias_test(strand_counts,
+                                 type = c("snv", "dbs"))
> strand_bias

$snv
       group mutation type transcribed untranscribed total     ratio     p_poisson significant
1      colon      snv  C>A        3893          2543  6436 1.5308691  6.575651e-64           *
2      colon      snv  C>G        1245          1174  2419 1.0604770  1.546502e-01
3      colon      snv  C>T        2813          2847  5660 0.9880576  6.609280e-01
4      colon      snv  T>A        1265           917  2182 1.3794984  9.765898e-14           *
5      colon      snv  T>C        2143          1511  3654 1.4182660  1.255387e-25           *
6      colon      snv  T>G         481           541  1022 0.8890943  6.490447e-02
7  intestine      snv  C>A        5582          3261  8843 1.7117449 6.028116e-136           *
8  intestine      snv  C>G        1284          1051  2335 1.2216936  1.548027e-06           *
9  intestine      snv  C>T        6076          7856 13932 0.7734216  1.815691e-51           *
10 intestine      snv  T>A        1658          1257  2915 1.3190135  1.169706e-13           *
11 intestine      snv  T>C        1985          1648  3633 1.2044903  2.427884e-08           *
12 intestine      snv  T>G         538           610  1148 0.8819672  3.608085e-02           *
13     liver      snv  C>A        1607          1427  3034 1.1261388  1.151742e-03           *
14     liver      snv  C>G         649           740  1389 0.8770270  1.571113e-02           *
15     liver      snv  C>T        7544          7477 15021 1.0089608  5.902258e-01
16     liver      snv  T>A        1628          1541  3169 1.0564568  1.265744e-01
17     liver      snv  T>C        4109          4518  8627 0.9094732  1.115587e-05           *
18     liver      snv  T>G         954          1236  2190 0.7718447  1.825034e-09           *

$dbs
       group mutation type transcribed untranscribed total     ratio    p_poisson significant
1      colon      dbs   AC           4             3     7 1.3333333 1.0000000000
2      colon      dbs   AT           1             1     2 1.0000000 1.0000000000
3      colon      dbs   CC          94            49   143 1.9183673 0.0002094705           *
4      colon      dbs   CG           2             1     3 2.0000000 1.0000000000
5      colon      dbs   CT          11             8    19 1.3750000 0.6476058960
6      colon      dbs   GC          12             4    16 3.0000000 0.0768127441
7      colon      dbs   TA           1             5     6 0.2000000 0.2187500000
8      colon      dbs   TC          11             8    19 1.3750000 0.6476058960
9      colon      dbs   TG          16            13    29 1.2307692 0.7110711038
10     colon      dbs   TT           0             3     3 0.0000000 0.2500000000
11 intestine      dbs   AC           7             9    16 0.7777778 0.8036193848
12 intestine      dbs   AT           1             2     3 0.5000000 1.0000000000
13 intestine      dbs   CC         216           178   394 1.2134831 0.0621798911
14 intestine      dbs   CG           3             2     5 1.5000000 1.0000000000
15 intestine      dbs   CT          21            19    40 1.1052632 0.8746293124
16 intestine      dbs   GC          13             7    20 1.8571429 0.2631759644
17 intestine      dbs   TA           1             4     5 0.2500000 0.3750000000
18 intestine      dbs   TC          15            11    26 1.3636364 0.5571970940
19 intestine      dbs   TG          17            20    37 0.8500000 0.7428293587
20 intestine      dbs   TT           7             7    14 1.0000000 1.0000000000
21     liver      dbs   AC          16            12    28 1.3333333 0.5715881884
```

```
22     liver      dbs   AT           7            6    13 1.1666667 1.0000000000
23     liver      dbs   CC          10            5    15 2.0000000 0.3017578125
24     liver      dbs   CG           2            6     8 0.3333333 0.2890625000
25     liver      dbs   CT          24           14    38 1.7142857 0.1433066543
26     liver      dbs   GC           3            2     5 1.5000000 1.0000000000
27     liver      dbs   TA           7            3    10 2.3333333 0.3437500000
28     liver      dbs   TC          21           10    31 2.1000000 0.0707555460
29     liver      dbs   TG          23           32    55 0.7187500 0.2806097177
30     liver      dbs   TT          25           22    47 1.1363636 0.7708669946
```

Plot the mutation spectrum with strand distinction:

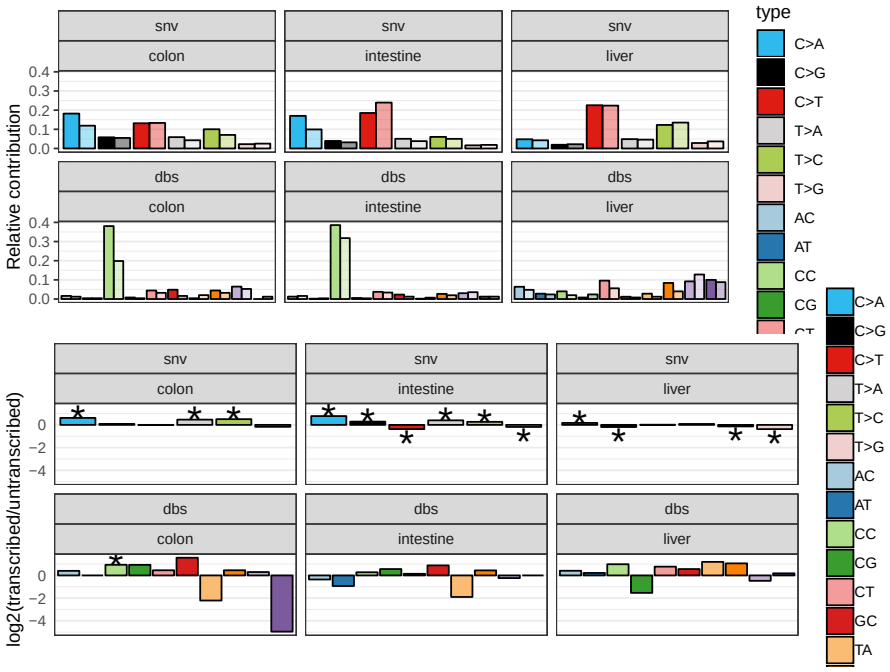
```
> ps1 <- plot_strand(strand_counts, mode = "relative")
```

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/ReplicationDirectionRegionsHaradhvala.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$Stop),
+   strand_info = repli_strand$Class)
> # UCSC seqlevelsstyle
> seqlevelsStyle(repli_strand_granges) = "UCSC"
> repli_strand_granges

GRanges object with 223 ranges and 1 metadata column:
        seqnames               ranges strand | strand_info
           <Rle>            <IRanges>  <Rle> |    <factor>
    [1]     chr1     [400001, 420000]      * |        left
    [2]     chr1     [420001, 440000]      * |        left
    [3]     chr1     [440001, 460000]      * |        left
    [4]     chr1     [460001, 480000]      * |        left
    [5]     chr1     [480001, 500000]      * |        left
    ...      ...                  ...    ... .         ...
  [219]     chr9 [67240001, 67260000]      * |       right
  [220]     chr9 [67260001, 67280000]      * |       right
  [221]     chr9 [67280001, 67300000]      * |       right
  [222]     chr9 [67300001, 67320000]      * |       right
  [223]     chr9 [79300001, 79320000]      * |       right
  -------
  seqinfo: 21 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[[1]], repli_strand_granges, mode = "replication",
+                          type = "all")
> lapply(strand_rep, head, 10)

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

$dbs
 [1] - - - - - - - - - -
```

```
Levels: left right -

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

Make mutation count matrices with transcriptional strand information.

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

$snv
              colon1 colon2 colon3 intestine1 intestine2
A[C>A]A-left       0      0      0          1          0
A[C>A]A-right      2      0      0          2          1
A[C>A]C-left       0      0      0          0          0
A[C>A]C-right      0      0      0          0          0
A[C>A]G-left       0      0      0          0          0

$dbs
            colon1 colon2 colon3 intestine1 intestine2
AC>CA-left       0      0      0          0          0
AC>CA-right      0      0      0          0          0
AC>CG-left       0      0      0          0          0
AC>CG-right      0      0      0          0          0
AC>CT-left       0      0      0          0          0

$indel
                              colon1 colon2 colon3 intestine1 intestine2
del.1bp.homopol.C.len.1-left       0      0      0          0          0
del.1bp.homopol.C.len.1-right      0      0      0          0          0
del.1bp.homopol.C.len.2-left       0      0      0          0          0
del.1bp.homopol.C.len.2-right      0      0      0          1          0
del.1bp.homopol.C.len.3-left       0      0      0          0          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, ref_genome, repli_strand_granges,
+                                       mode = "replication",
+                                       type = "all")
> lapply(mut_mat_s_rep2, function(x) x[1:5, 1:5])

$snv
              colon1 colon2 colon3 intestine1 intestine2
A[C>A]A-right      2      0      0          2          1
A[C>A]A-left       0      0      0          1          0
```

```
A[C>A]C-right      0      0      0          0          0
A[C>A]C-left       0      0      0          0          0
A[C>A]G-right      0      1      0          0          0

$dbs
            colon1 colon2 colon3 intestine1 intestine2
AC>CA-right      0      0      0          0          0
AC>CA-left       0      0      0          0          0
AC>CG-right      0      0      0          0          0
AC>CG-left       0      0      0          0          0
AC>CT-right      0      0      0          0          0

$indel
                              colon1 colon2 colon3 intestine1 intestine2
del.1bp.homopol.C.len.1-right      0      0      0          0          0
del.1bp.homopol.C.len.1-left       0      0      0          0          0
del.1bp.homopol.C.len.2-right      0      0      0          1          0
del.1bp.homopol.C.len.2-left       0      0      0          0          0
del.1bp.homopol.C.len.3-right      0      0      0          0          0
```

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

```
> strand_counts_rep <- strand_occurrences(mut_mat_s_rep, by=tissue,
+                                         type = "all")
> lapply(strand_counts_rep, head)

$snv
   group mutation type strand no_mutations relative_contribution
1  colon      snv  C>A   left           4            0.08333333
4  colon      snv  C>A  right          13            0.27083333
7  colon      snv  C>G   left           1            0.02083333
10 colon      snv  C>G  right           5            0.10416667
13 colon      snv  C>T   left           1            0.02083333
16 colon      snv  C>T  right          10            0.20833333

$dbs
   group mutation type strand no_mutations relative_contribution
1  colon      dbs   AC   left           0                   NaN
4  colon      dbs   AC  right           0                   NaN
7  colon      dbs   AT   left           0                   NaN
10 colon      dbs   AT  right           0                   NaN
13 colon      dbs   CC   left           0                   NaN
16 colon      dbs   CC  right           0                   NaN

$indel
   group mutation              type strand no_mutations relative_contribution
1  colon    indel del.1bp.homopol.C   left           0                     0
4  colon    indel del.1bp.homopol.C  right           0                     0
7  colon    indel del.1bp.homopol.T   left           0                     0
10 colon    indel del.1bp.homopol.T  right           0                     0
13 colon    indel      del.mh.len.2   left           0                     0
16 colon    indel      del.mh.len.2  right           0                     0
```

Perform Poisson test for strand asymmetry significance testing:

```
> strand_bias_rep <- strand_bias_test(strand_counts_rep,
+                                    type = "all")
> strand_bias_rep

$snv
       group mutation type left right total      ratio    p_poisson significant
1      colon      snv  C>A    4    13    17 0.30769231 4.904175e-02           *
2      colon      snv  C>G    1     5     6 0.20000000 2.187500e-01
3      colon      snv  C>T    1    10    11 0.10000000 1.171875e-02           *
4      colon      snv  T>A    1     6     7 0.16666667 1.250000e-01
5      colon      snv  T>C    1     6     7 0.16666667 1.250000e-01
6      colon      snv  T>G    0     0     0        NaN 1.000000e+00
7  intestine      snv  C>A    6    24    30 0.25000000 1.430906e-03           *
8  intestine      snv  C>G    1    11    12 0.09090909 6.347656e-03           *
9  intestine      snv  C>T   12    45    57 0.26666667 1.312744e-05           *
10 intestine      snv  T>A    2     7     9 0.28571429 1.796875e-01
11 intestine      snv  T>C    2    17    19 0.11764706 7.286072e-04           *
12 intestine      snv  T>G    1     4     5 0.25000000 3.750000e-01
13     liver      snv  C>A    3    12    15 0.25000000 3.515625e-02           *
14     liver      snv  C>G    1     3     4 0.33333333 6.250000e-01
15     liver      snv  C>T    8    51    59 0.15686275 9.052391e-09           *
16     liver      snv  T>A    2    11    13 0.18181818 2.246094e-02           *
17     liver      snv  T>C    8    20    28 0.40000000 3.569814e-02           *
18     liver      snv  T>G    0     5     5 0.00000000 6.250000e-02

$dbs
       group mutation type left right total ratio p_poisson significant
1      colon      dbs   AC    0     0     0   NaN       1.0
2      colon      dbs   AT    0     0     0   NaN       1.0
3      colon      dbs   CC    0     0     0   NaN       1.0
4      colon      dbs   CG    0     0     0   NaN       1.0
5      colon      dbs   CT    0     0     0   NaN       1.0
6      colon      dbs   GC    0     0     0   NaN       1.0
7      colon      dbs   TA    0     0     0   NaN       1.0
8      colon      dbs   TC    0     0     0   NaN       1.0
9      colon      dbs   TG    0     0     0   NaN       1.0
10     colon      dbs   TT    0     0     0   NaN       1.0
11 intestine      dbs   AC    0     0     0   NaN       1.0
12 intestine      dbs   AT    0     0     0   NaN       1.0
13 intestine      dbs   CC    0     1     1     0       1.0
14 intestine      dbs   CG    0     1     1     0       1.0
15 intestine      dbs   CT    0     0     0   NaN       1.0
16 intestine      dbs   GC    0     0     0   NaN       1.0
17 intestine      dbs   TA    0     0     0   NaN       1.0
18 intestine      dbs   TC    0     0     0   NaN       1.0
19 intestine      dbs   TG    0     0     0   NaN       1.0
20 intestine      dbs   TT    0     0     0   NaN       1.0
21     liver      dbs   AC    0     1     1     0       1.0
```

```
22     liver      dbs   AT    0     0     0   NaN       1.0
23     liver      dbs   CC    0     0     0   NaN       1.0
24     liver      dbs   CG    0     0     0   NaN       1.0
25     liver      dbs   CT    0     0     0   NaN       1.0
26     liver      dbs   GC    0     0     0   NaN       1.0
27     liver      dbs   TA    0     0     0   NaN       1.0
28     liver      dbs   TC    0     0     0   NaN       1.0
29     liver      dbs   TG    0     2     2     0       0.5
30     liver      dbs   TT    0     1     1     0       1.0

$indel
       group mutation              type left right total     ratio    p_poisson significant
1      colon    indel del.1bp.homopol.C    0     0     0       NaN 1.000000e+00
2      colon    indel del.1bp.homopol.T    0     0     0       NaN 1.000000e+00
3      colon    indel      del.mh.len.2    0     0     0       NaN 1.000000e+00
4      colon    indel      del.mh.len.3    0     0     0       NaN 1.000000e+00
5      colon    indel      del.mh.len.4    0     0     0       NaN 1.000000e+00
6      colon    indel     del.mh.len.5+    0     1     1 0.0000000 1.000000e+00
7      colon    indel     del.rep.len.2    0     0     0       NaN 1.000000e+00
8      colon    indel     del.rep.len.3    0     0     0       NaN 1.000000e+00
9      colon    indel     del.rep.len.4    0     0     0       NaN 1.000000e+00
10     colon    indel    del.rep.len.5+    0     0     0       NaN 1.000000e+00
11     colon    indel ins.1bp.homopol.C    0     0     0       NaN 1.000000e+00
12     colon    indel ins.1bp.homopol.T    0     3     3 0.0000000 2.500000e-01
13     colon    indel     ins.rep.len.2    0     0     0       NaN 1.000000e+00
14     colon    indel     ins.rep.len.3    0     0     0       NaN 1.000000e+00
15     colon    indel     ins.rep.len.4    0     0     0       NaN 1.000000e+00
16     colon    indel    ins.rep.len.5+    0     0     0       NaN 1.000000e+00
17 intestine    indel del.1bp.homopol.C    0     1     1 0.0000000 1.000000e+00
18 intestine    indel del.1bp.homopol.T    1     1     2 1.0000000 1.000000e+00
19 intestine    indel      del.mh.len.2    0     1     1 0.0000000 1.000000e+00
20 intestine    indel      del.mh.len.3    0     0     0       NaN 1.000000e+00
21 intestine    indel      del.mh.len.4    0     0     0       NaN 1.000000e+00
22 intestine    indel     del.mh.len.5+    0     0     0       NaN 1.000000e+00
23 intestine    indel     del.rep.len.2    0     2     2 0.0000000 5.000000e-01
24 intestine    indel     del.rep.len.3    0     0     0       NaN 1.000000e+00
25 intestine    indel     del.rep.len.4    0     0     0       NaN 1.000000e+00
26 intestine    indel    del.rep.len.5+    0     0     0       NaN 1.000000e+00
27 intestine    indel ins.1bp.homopol.C    0     1     1 0.0000000 1.000000e+00
28 intestine    indel ins.1bp.homopol.T    1     3     4 0.3333333 6.250000e-01
29 intestine    indel     ins.rep.len.2    0     0     0       NaN 1.000000e+00
30 intestine    indel     ins.rep.len.3    0     0     0       NaN 1.000000e+00
31 intestine    indel     ins.rep.len.4    0     0     0       NaN 1.000000e+00
32 intestine    indel    ins.rep.len.5+    0     0     0       NaN 1.000000e+00
33     liver    indel del.1bp.homopol.C    3     3     6 1.0000000 1.000000e+00
34     liver    indel del.1bp.homopol.T   13   123   136 0.1056911 1.239373e-23           *
35     liver    indel      del.mh.len.2    0     2     2 0.0000000 5.000000e-01
36     liver    indel      del.mh.len.3    0     0     0       NaN 1.000000e+00
37     liver    indel      del.mh.len.4    0     0     0       NaN 1.000000e+00
38     liver    indel     del.mh.len.5+    0     0     0       NaN 1.000000e+00
39     liver    indel     del.rep.len.2    1    10    11 0.1000000 1.171875e-02           *
```

```
40     liver    indel     del.rep.len.3    0     3     3 0.0000000 2.500000e-01
41     liver    indel     del.rep.len.4    0     1     1 0.0000000 1.000000e+00
42     liver    indel    del.rep.len.5+    0     1     1 0.0000000 1.000000e+00
43     liver    indel ins.1bp.homopol.C    1     2     3 0.5000000 1.000000e+00
44     liver    indel ins.1bp.homopol.T    3     8    11 0.3750000 2.265625e-01
45     liver    indel     ins.rep.len.2    0     6     6 0.0000000 3.125000e-02 *
46     liver    indel     ins.rep.len.3    0     0     0       NaN 1.000000e+00
47     liver    indel     ins.rep.len.4    0     0     0       NaN 1.000000e+00
48     liver    indel    ins.rep.len.5+    0     0     0       NaN 1.000000e+00
```

Plot the mutation spectrum with strand distinction:

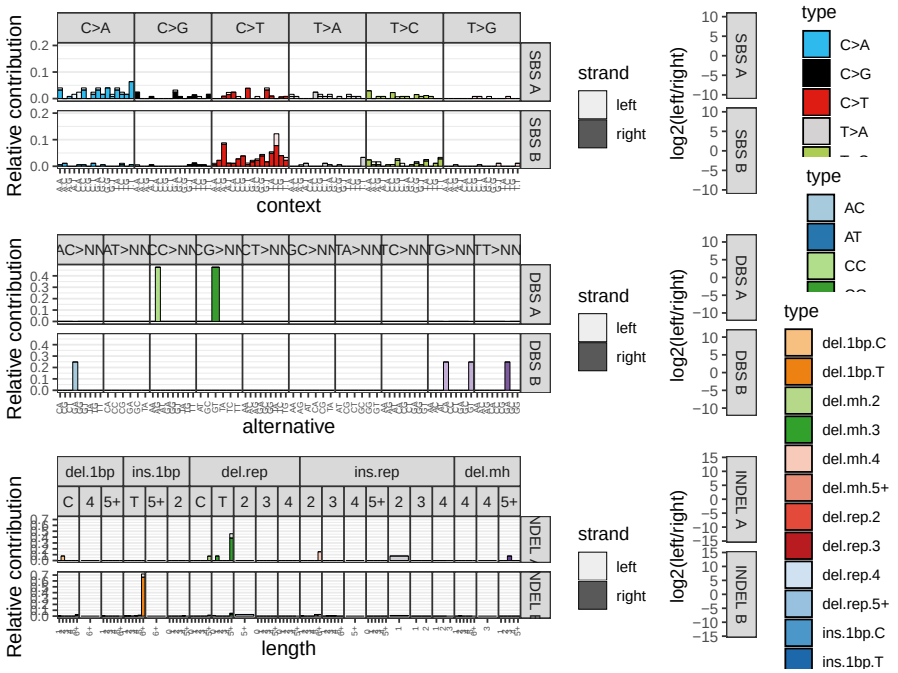
```
> ps1 <- plot_strand(strand_counts_rep, mode = "relative")
```

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 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, type = "all", 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:

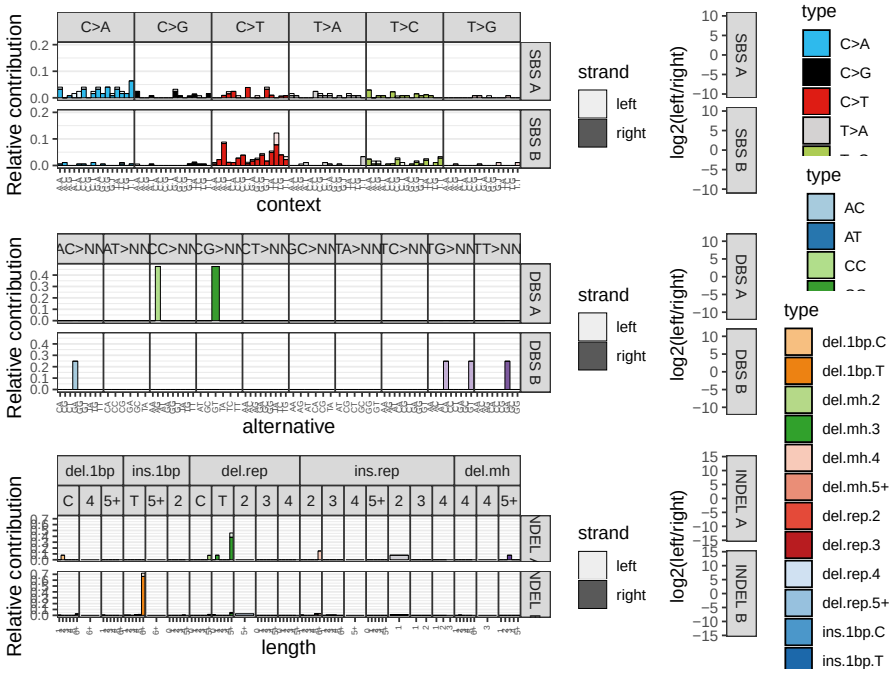
```
> a <- plot_strand_profiles(nmf_res_strand$signatures, condensed = TRUE,
+                           mode = "replication",
+                           type = "all")
```

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

```
> b <- plot_signature_strand_bias(nmf_res_strand$signatures,
+                                 type = "all")
```

Combine the plots into one figure:

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



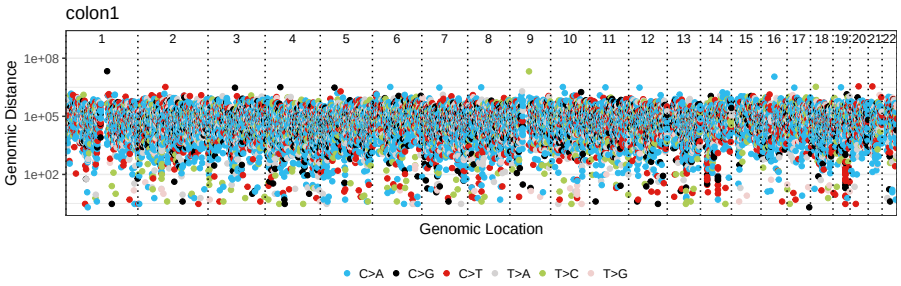
6 Genomic distribution

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.

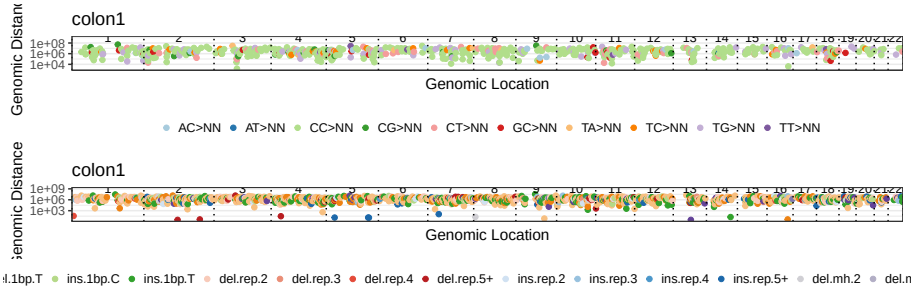
Make rainfall plot of single base substitutions from sample 1 over all autosomal chromosomes

```
> # 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[[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)
```

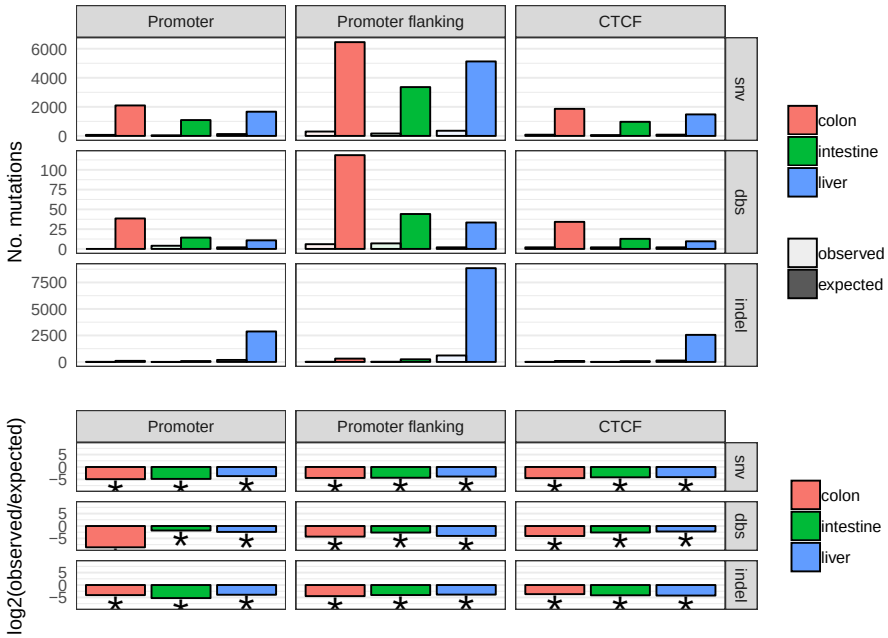
Test for enrichment or depletion of mutations in your defined genomic regions using a binomial test. For this test, the chance of observing a mutation is calculated as the total number of mutations, divided by the total number of surveyed bases.

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

```
> ## 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      dbs   1955       727070334               14327310        0
2 intestine          Promoter      dbs    728       727070334               14327310        4
3     liver          Promoter      dbs    551       727070334               14327310        2
4     colon Promoter flanking      dbs   1955       727070334               44087613        6
5 intestine Promoter flanking      dbs    728       727070334               44087613        7
6     liver Promoter flanking      dbs    551       727070334               44087613        2
          prob  expected    effect         pval significant
1 2.688873e-06  38.52432 depletion 1.858136e-17           *
2 1.001279e-06  14.34563 depletion 1.397752e-03           *
3 7.578359e-07  10.85775 depletion 1.363296e-03           *
4 2.688873e-06 118.54600 depletion 1.331698e-42           *
5 1.001279e-06  44.14399 depletion 5.165193e-12           *
6 7.578359e-07  33.41118 depletion 1.829959e-12           *
```

```
> 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

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 3.1.1, 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 2.0.1, digest 0.6.18, dplyr 0.7.8, evaluate 0.14, fs 1.2.6, 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 1.0.2, pkgconfig 2.0.2, pkgload 1.0.2, plyr 1.8.4, pracma 2.2.2, prettyunits 1.0.2, processx 3.2.1, 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 2.0.2, reshape2 1.4.3, rlang 0.3.1, 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 1.4.0, 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