Introduction to *MutationalPatterns*

***Francis Blokzijl*[1], *Roel Janssen*[1], *Ruben van Boxtel*[1], and *Edwin Cuppen*[1]**

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

October 29, 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, of which a preprint is available at bioRxiv: https://doi.org/10.1101/071761

2 Data

To perform the mutational pattern analyses, you need to load one or multiple VCF files with single-nucleotide variant 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.Aofficinalis.NCBI.V1"          "BSgenome.Athaliana.TAIR.04232008"
```

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)
```

```
[1] "GRangesList object of length 9 with 0 metadata columns"
```

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 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 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	37	90	22	209	136
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	146	93	276	103	209	73	48	228
liver3	39	28	62	15	32	24	7	55

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:

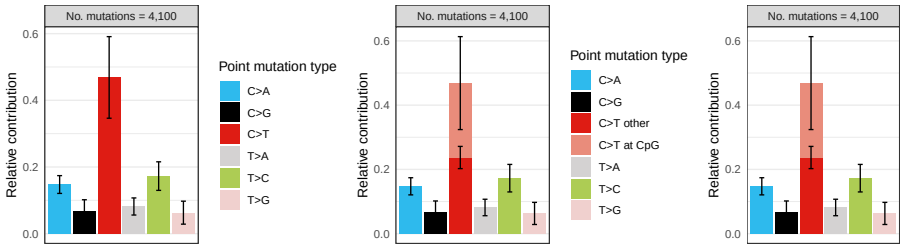
```
> p2 <- plot_spectrum(type_occurrences, CT = TRUE)
```

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))
```



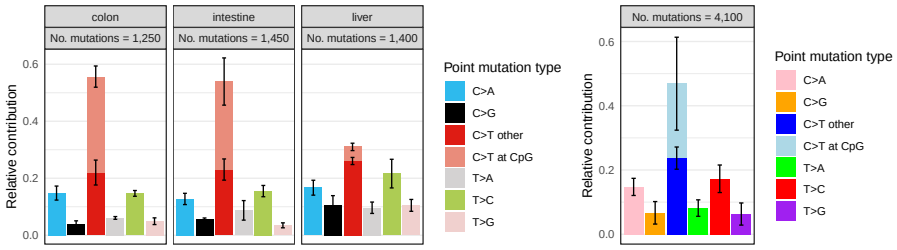
You can facet the per sample group, e.g. plot the spectrum for each tissue separately:

```
> 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 96 mutational profile

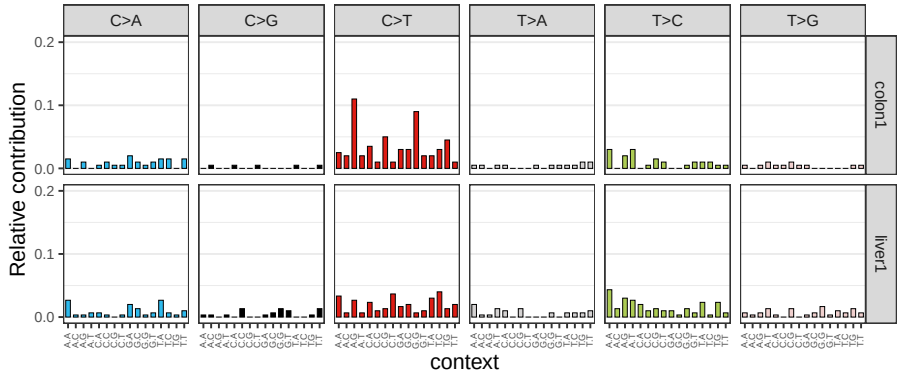
Make a 96 trinucleodide mutation count matrix:

```
> 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      3     10     10          5         19          6      8     10      3
A[C>A]C      0      3      3          1          8          4      1      8      2
A[C>A]G      2      3      3          1          4          0      1      6      2
A[C>A]T      0      2      9          0          9          2      2     12      2
C[C>A]A      1      8      5          0          8          7      2     15      3
C[C>A]C      2      5      3          1          3          2      1     16      2
```

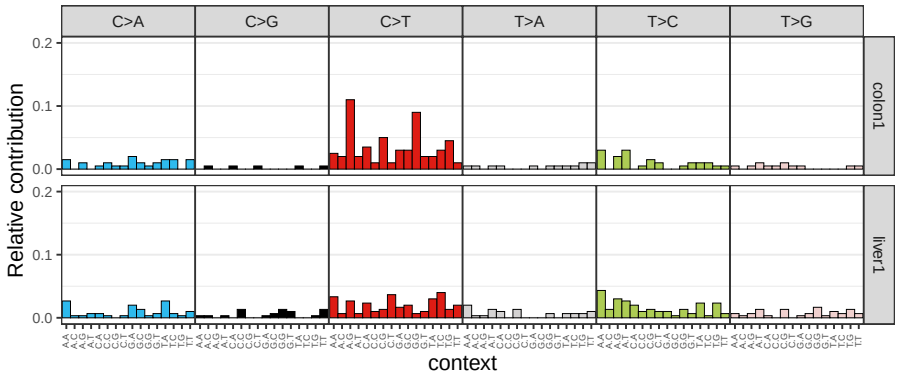
Plot the 96 profile of two samples:

```
> plot_96_profile(mut_mat[,c(1,7)])
```



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

```
> plot_96_profile(mut_mat[,c(1,7)], condensed = TRUE)
```



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 base substitution types with a certain sequence context. 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."

First add a small psuedocount to your mutation count matrix:

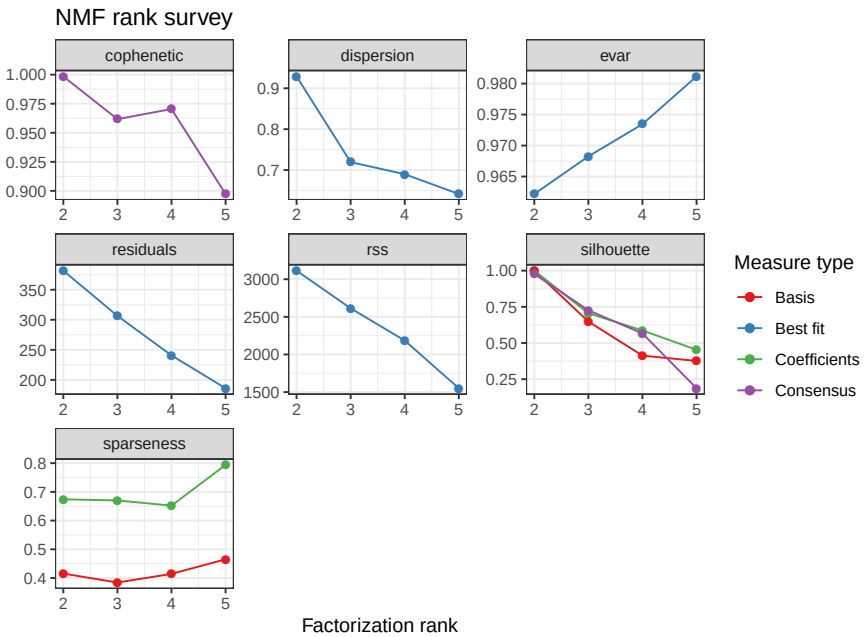
```
> mut_mat <- mut_mat + 0.0001
```

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):

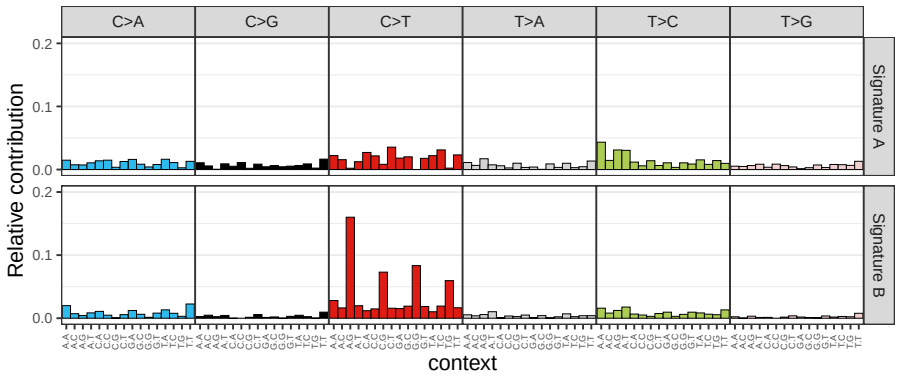
```
> nmf_res <- extract_signatures(mut_mat, rank = 2, nrun = 10)
```

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_96_profile(nmf_res$signatures, condensed = TRUE)
```



Visualize the contribution of the signatures in a barplot:

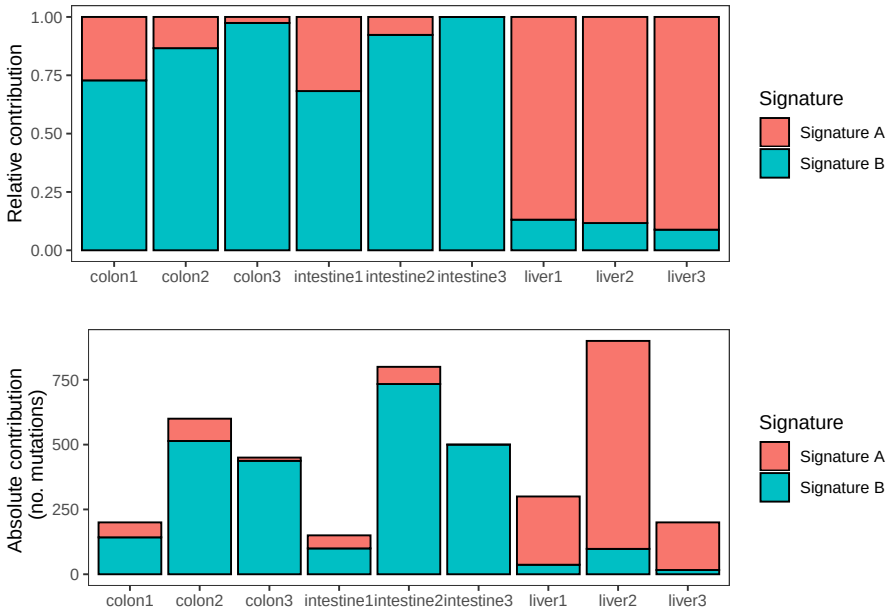
```
> pc1 <- plot_contribution(nmf_res$contribution, nmf_res$signature,
+                          mode = "relative")
```

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

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

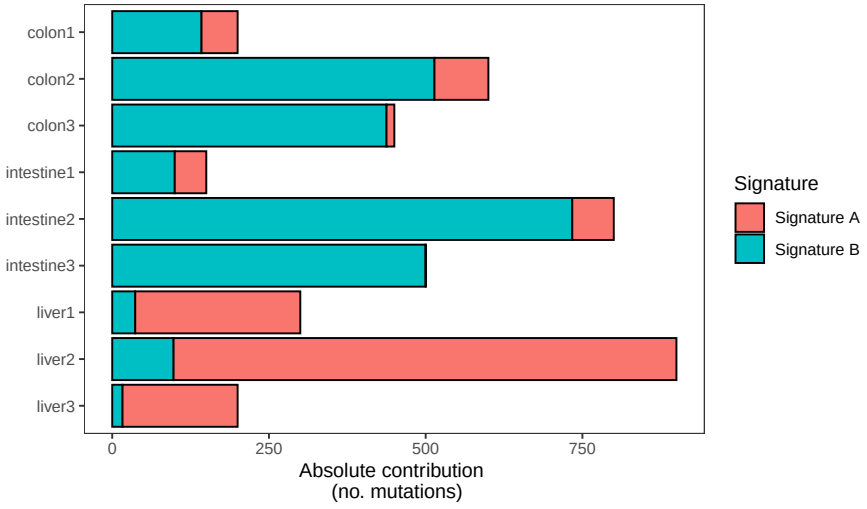
Combine the two plots:

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



Flip X and Y coordinates:

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



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

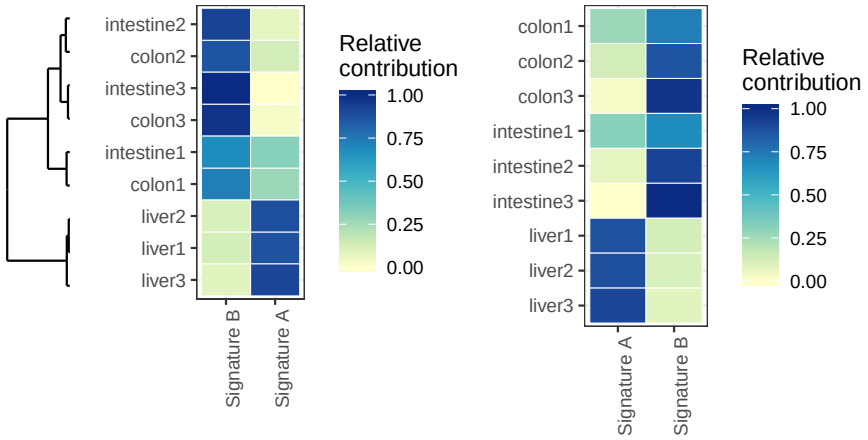
```
> pch1 <- plot_contribution_heatmap(nmf_res$contribution,
+                                   sig_order = c("Signature B", "Signature A"))
```

Plot signature contribution as a heatmap without sample clustering:

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

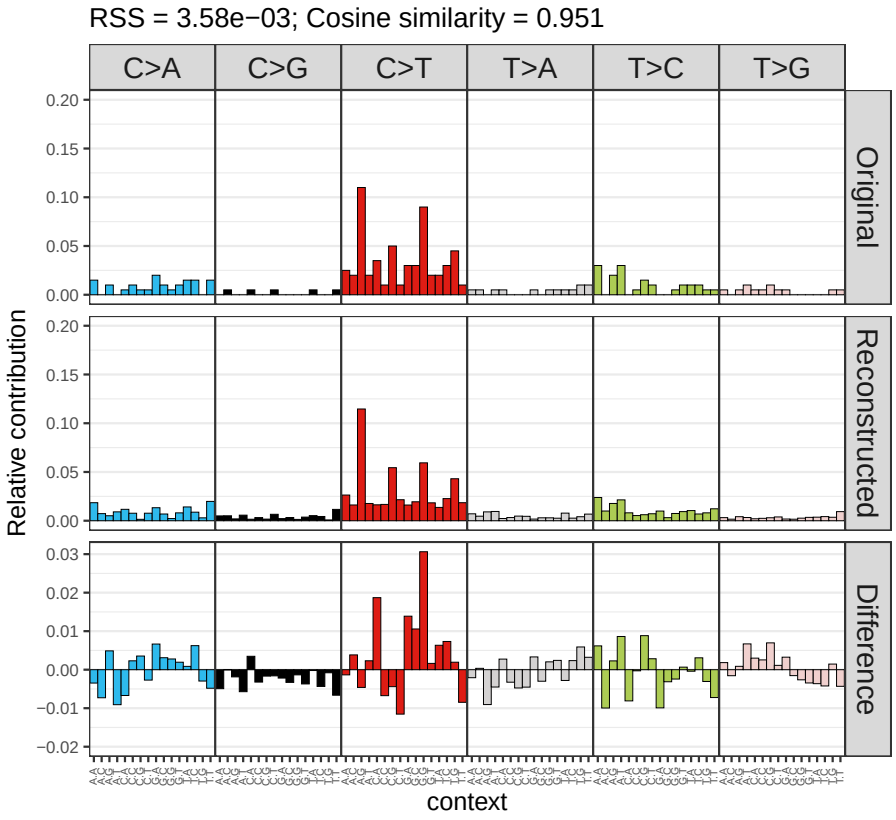
Combine the plots into one figure:

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



Compare the reconstructed mutational profile with the original mutational profile:

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



4.2 Find optimal contribution of known signatures

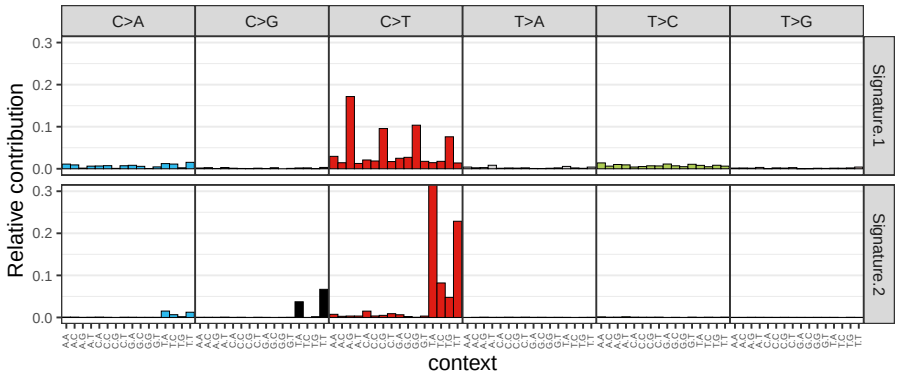
4.2.1 COSMIC mutational signatures

Download mutational signatures from the COSMIC website:

```
> sp_url <- paste("https://cancer.sanger.ac.uk/cancergenome/assets/",
+                 "signatures_probabilities.txt", sep = "")
> cancer_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), cancer_signatures$Somatic.Mutation.Type)
> # Reorder cancer signatures dataframe
> cancer_signatures = cancer_signatures[as.vector(new_order),]
> # Add trinucletiode changes names as row.names
> row.names(cancer_signatures) = cancer_signatures$Somatic.Mutation.Type
> # Keep only 96 contributions of the signatures in matrix
> cancer_signatures = as.matrix(cancer_signatures[,4:33])
```

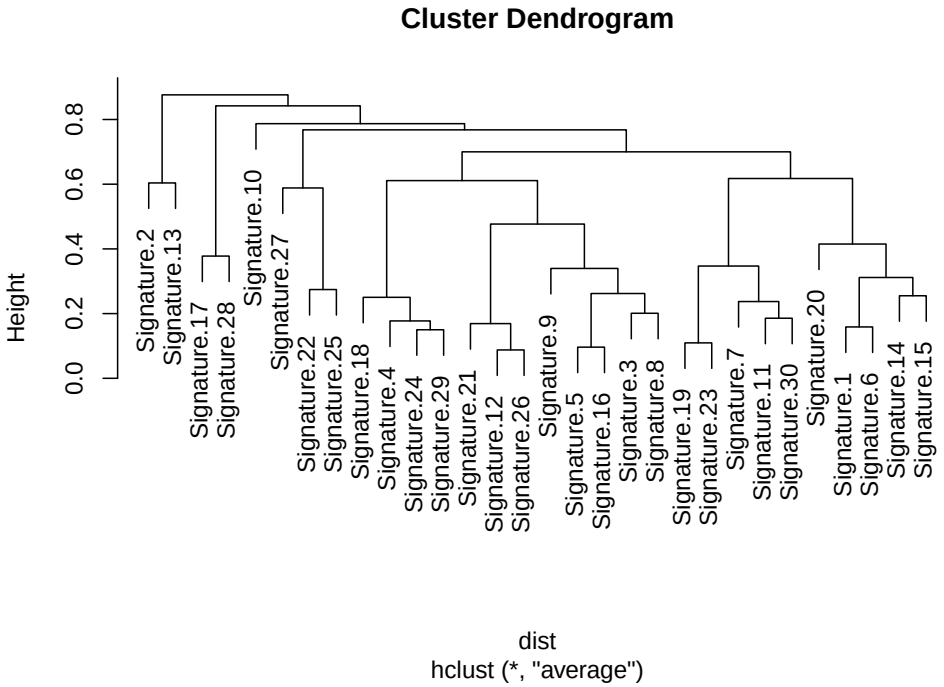
Plot mutational profile of the first two COSMIC signatures:

```
> plot_96_profile(cancer_signatures[,1:2], condensed = TRUE, ymax = 0.3)
```



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

```
> hclust_cosmic = cluster_signatures(cancer_signatures, method = "average")
> # store signatures in new order
> cosmic_order = colnames(cancer_signatures)[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 ad-

vantage 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`.

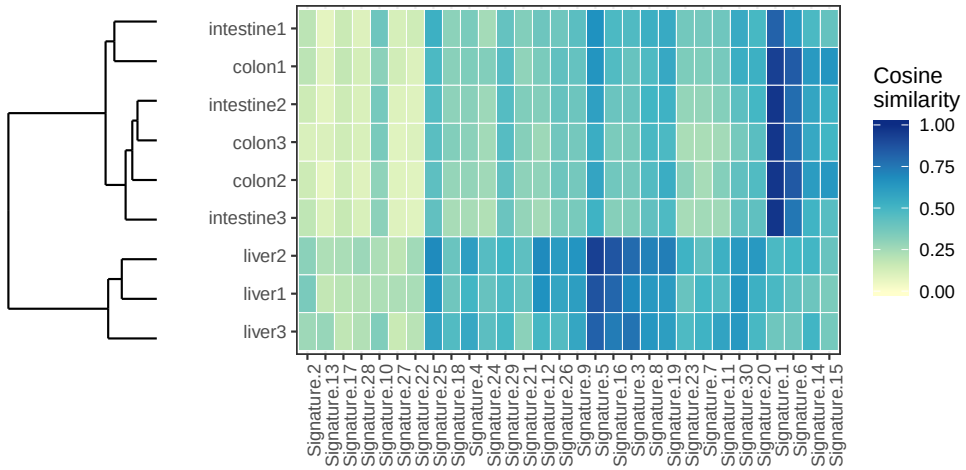
The cosine similarity between two mutational profiles/signatures can be calculated with `cos_sim`:

```
> cos_sim(mut_mat[,1], cancer_signatures[,1])

[1] 0.9319384
```

Calculate pairwise cosine similarity between mutational profiles and COSMIC signatures:

```
> cos_sim_samples_signatures = cos_sim_matrix(mut_mat, cancer_signatures)
> # Plot heatmap with specified signature order
> plot_cosine_heatmap(cos_sim_samples_signatures,
+                     col_order = cosmic_order,
+                     cluster_rows = TRUE)
```



4.2.3 Find optimal contribution of COSMIC signatures to reconstruct 96 mutational profiles

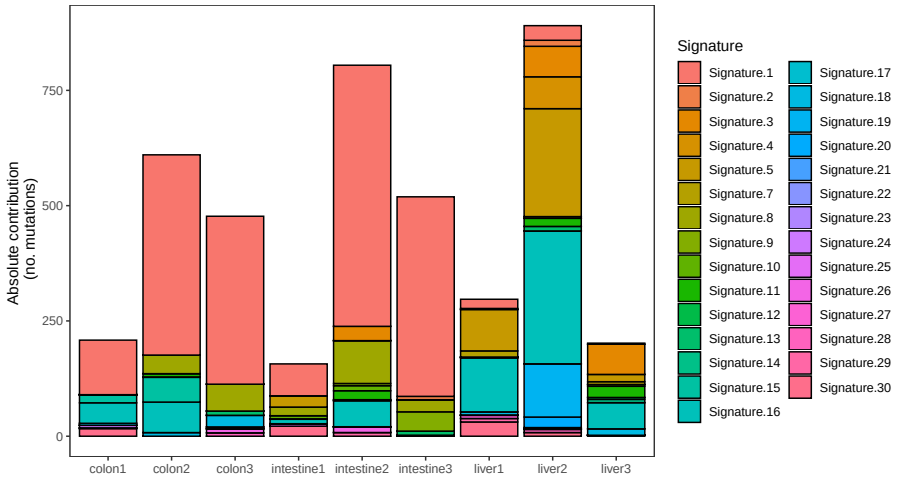
In addition to *de novo* extraction of signatures, the contribution of any set of signatures to the mutational profile of a sample can be quantified. This unique feature is specifically useful for mutational signature analyses of small cohorts or individual samples, but also to relate own findings to known signatures and published findings. The `fit_to_signatures` function finds the optimal linear combination of mutational signatures that most closely reconstructs the mutation matrix by solving a non-negative least-squares constraints problem.

Fit mutation matrix to the COSMIC mutational signatures:

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

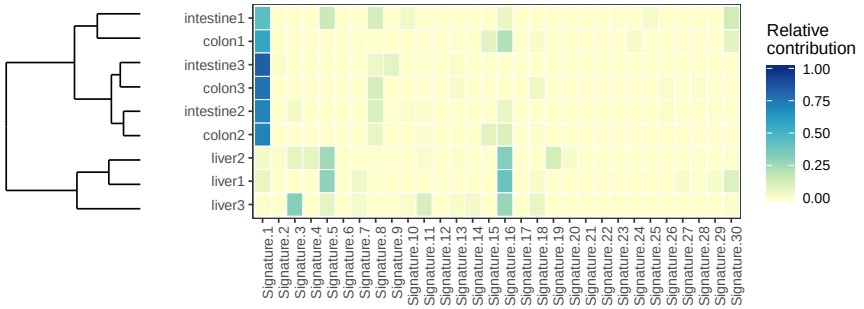
Plot the optimal contribution of the COSMIC signatures in each sample as a stacked barplot.

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



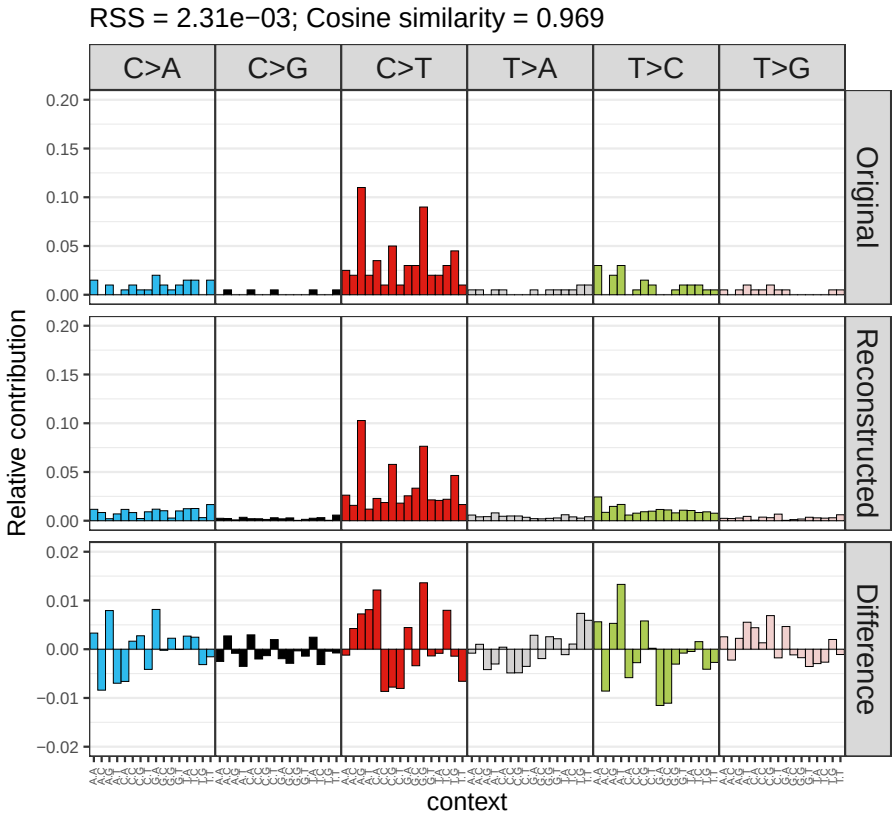
Plot relative contribution of the cancer signatures in each sample as a heatmap with sample clustering:

```
> 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[,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`:

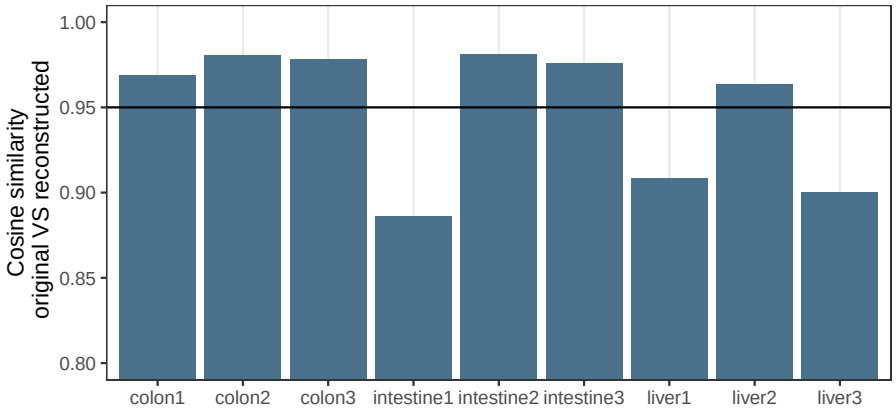
```
> # calculate all pairwise cosine similarities
> cos_sim_ori_rec <- cos_sim_matrix(mut_mat, fit_res$reconstructed)
> # extract cosine similarities per sample between original and reconstructed
> cos_sim_ori_rec <- as.data.frame(diag(cos_sim_ori_rec))
```

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. 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.

Get gene definitions for your reference genome:

```
> # For example get known genes table from UCSC for hg19 using
> # BiocManager::install("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 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)
> 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). NB: only those mutations that are located within gene bodies are counted.

```
> mut_mat_s <- mut_matrix_stranded(vcfs, 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)

   group type        strand no_mutations relative_contribution
1  colon  C>A   transcribed           32            0.07289294
4  colon  C>A untranscribed           23            0.05239180
7  colon  C>G   transcribed           11            0.02505695
10 colon  C>G untranscribed           10            0.02277904
13 colon  C>T   transcribed          135            0.30751708
```

```
16 colon  C>T untranscribed          115            0.26195900
```

Perform Poisson test for strand asymmetry significance testing:

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

```
       group type transcribed untranscribed total     ratio  p_poisson significant
1      colon  C>A          32            23    55 1.3913043 0.28060972
2      colon  C>G          11            10    21 1.1000000 1.00000000
3      colon  C>T         135           115   250 1.1739130 0.22942486
4      colon  T>A          12             9    21 1.3333333 0.66362381
5      colon  T>C          36            32    68 1.1250000 0.71630076
6      colon  T>G          15             9    24 1.6666667 0.30745625
7  intestine  C>A          34            27    61 1.2592593 0.44262600
8  intestine  C>G          18            21    39 0.8571429 0.74925862
9  intestine  C>T         144           129   273 1.1162791 0.39685899
10 intestine  T>A          23            18    41 1.2777778 0.53270926
11 intestine  T>C          52            38    90 1.3684211 0.17024240
12 intestine  T>G          10            10    20 1.0000000 1.00000000
13     liver  C>A          45            44    89 1.0227273 1.00000000
14     liver  C>G          19            34    53 0.5588235 0.05343881
15     liver  C>T          87            82   169 1.0609756 0.75842199
16     liver  T>A          36            23    59 1.5652174 0.11747735
17     liver  T>C          75            52   127 1.4423077 0.05048701
18     liver  T>G          23            43    66 0.5348837 0.01865726           *
```

Plot the mutation spectrum with strand distinction:

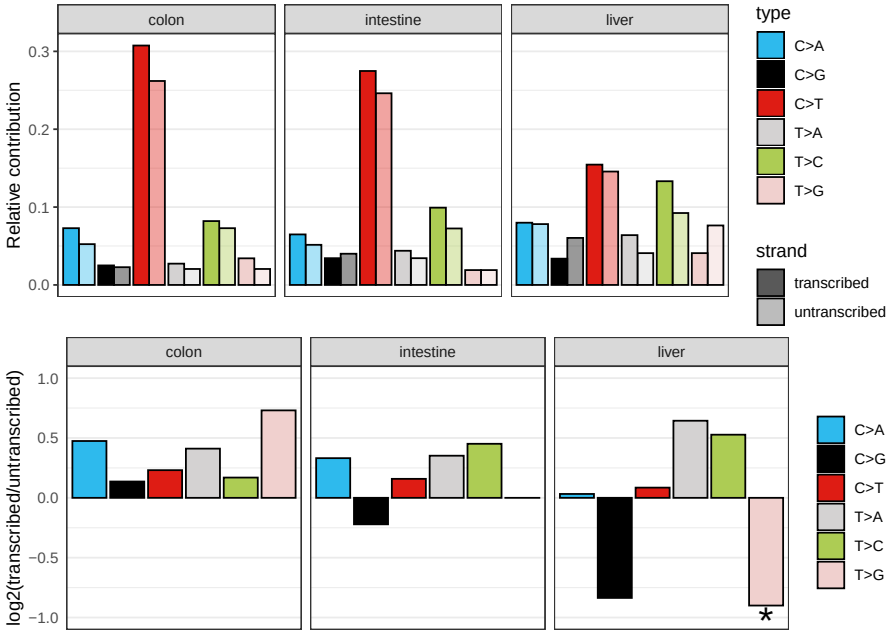
```
> ps1 <- plot_strand(strand_counts, mode = "relative")
```

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$Stop),
+   strand_info = repli_strand$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.

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

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

Make mutation count matrix with transcriptional strand information (96 trinucleotides * 2 strands = 192 features).

```
> mut_mat_s_rep <- mut_matrix_stranded(vcfs, 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, 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)
```

```
   group type        strand no_mutations relative_contribution
1  colon  C>A   transcribed           32            0.07289294
4  colon  C>A untranscribed           23            0.05239180
7  colon  C>G   transcribed           11            0.02505695
10 colon  C>G untranscribed           10            0.02277904
13 colon  C>T   transcribed          135            0.30751708
16 colon  C>T untranscribed          115            0.26195900
```

Perform Poisson test for strand asymmetry significance testing:

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

```
       group type left right total     ratio  p_poisson significant
1      colon  C>A   28    42    70 0.6666667 0.11960934
2      colon  C>G   12    12    24 1.0000000 1.00000000
3      colon  C>T  157   128   285 1.2265625 0.09702977
4      colon  T>A   12    10    22 1.2000000 0.83181190
5      colon  T>C   41    41    82 1.0000000 1.00000000
6      colon  T>G   16    11    27 1.4545455 0.44206834
7  intestine  C>A   31    33    64 0.9393939 0.90065325
8  intestine  C>G   19    11    30 1.7272727 0.20048842
9  intestine  C>T  146   162   308 0.9012346 0.39274995
10 intestine  T>A   21    15    36 1.4000000 0.40503225
11 intestine  T>C   45    34    79 1.3235294 0.26042553
12 intestine  T>G   10    11    21 0.9090909 1.00000000
13     liver  C>A   47    51    98 0.9215686 0.76203622
14     liver  C>G   34    33    67 1.0303030 1.00000000
15     liver  C>T  107    98   205 1.0918367 0.57644403
16     liver  T>A   24    31    55 0.7741935 0.41875419
17     liver  T>C   75    63   138 1.1904762 0.34911517
18     liver  T>G   29    34    63 0.8529412 0.61465502
```

Plot the mutation spectrum with strand distinction:

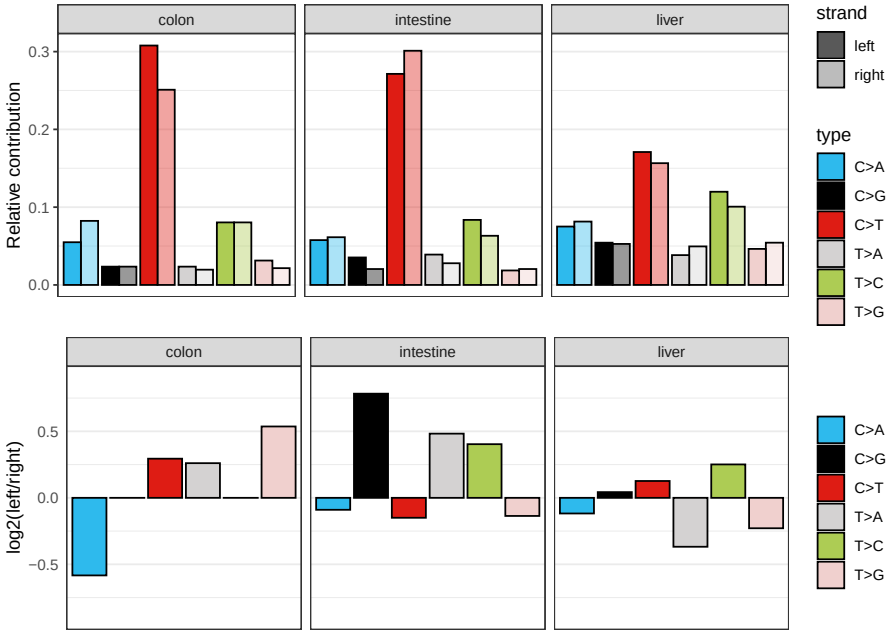
```
> ps1 <- plot_strand(strand_counts_rep, mode = "relative")
```

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 Extract signatures with strand bias

Extract 2 signatures from mutation count matrix with strand features:

```
> nmf_res_strand <- extract_signatures(mut_mat_s, rank = 2)
> # Provide signature names
> colnames(nmf_res_strand$signatures) <- c("Signature A", "Signature B")
```

Plot signatures with 192 features:

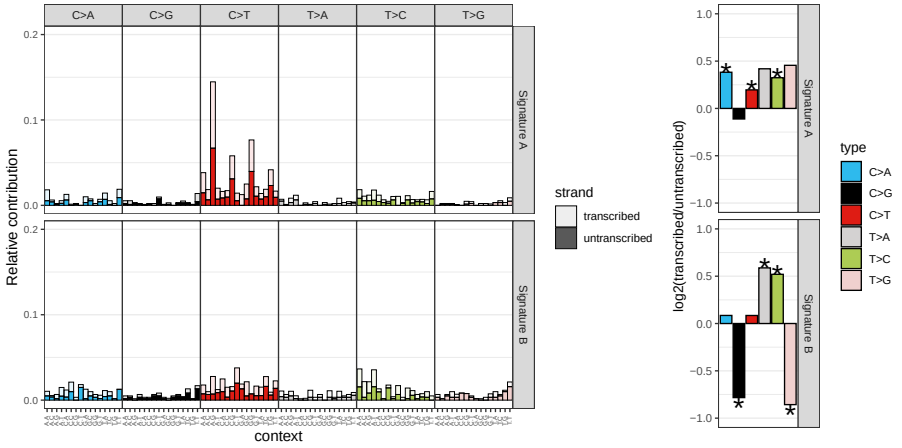
```
> a <- plot_192_profile(nmf_res_strand$signatures, condensed = TRUE)
```

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))
```



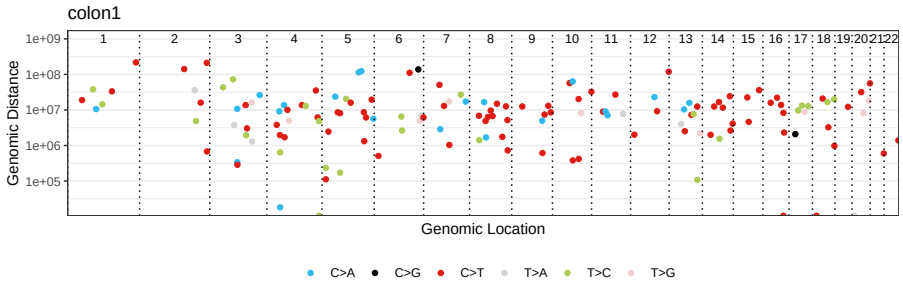
6 Genomic distribution

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 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)
```



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:

```
> if (!requireNamespace("BiocManager", quietly=TRUE))
+     install.packages("BiocManager")
> BiocManager::install("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)
```

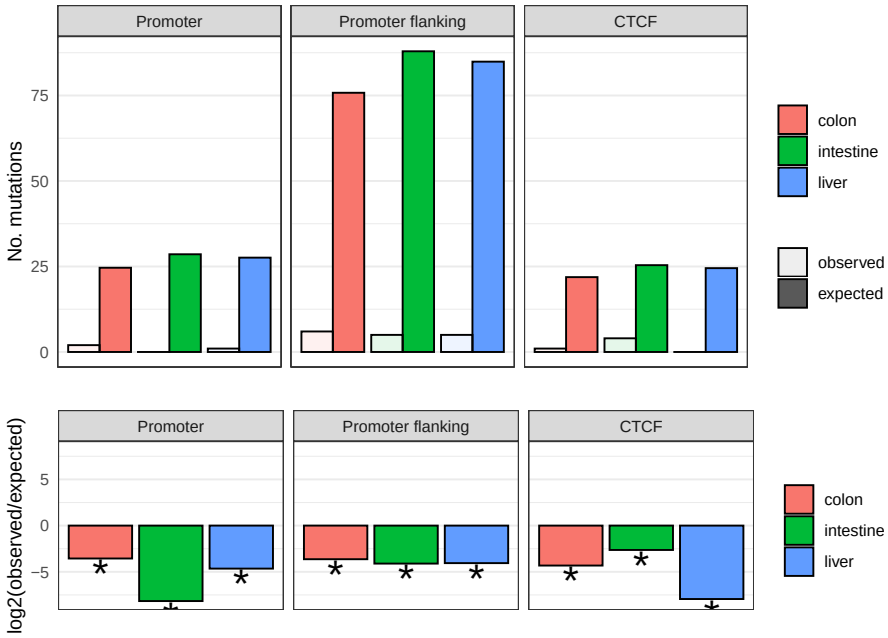
Test for enrichment or depletion of mutations in your defined genomic regions using a binomial test. For this test, the chance of observing a mutation is calculated as the total number of mutations, divided by the total number of surveyed bases.

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

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

```
         by            region n_muts surveyed_length surveyed_region_length observed
1     colon          Promoter   1250       727070334               14327310        2
2 intestine          Promoter   1450       727070334               14327310        0
3     liver          Promoter   1400       727070334               14327310        1
4     colon Promoter flanking   1250       727070334               44087613        6
5 intestine Promoter flanking   1450       727070334               44087613        5
6     liver Promoter flanking   1400       727070334               44087613        5
          prob expected    effect         pval significant
1 1.719228e-06 24.63192 depletion 6.602046e-09           *
2 1.994305e-06 28.57303 depletion 3.898344e-13           *
3 1.925536e-06 27.58775 depletion 2.985121e-11           *
4 1.719228e-06 75.79668 depletion 3.449547e-25           *
5 1.994305e-06 87.92415 depletion 3.030213e-31           *
6 1.925536e-06 84.89228 depletion 5.283377e-30           *
```

```
> 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

7 Session Information

- R version 3.6.1 (2019-07-05), `x86_64-apple-darwin15.6.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Running under: `OS X El Capitan 10.11.6`
- Matrix products: default
- BLAS: `/Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib`

- LAPACK:
  `/Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.48.0, BSgenome 1.54.0, BSgenome.Hsapiens.UCSC.hg19 1.4.0, Biobase 2.46.0, BiocGenerics 0.32.0, Biostrings 2.54.0, GenomeInfoDb 1.22.0, GenomicFeatures 1.38.0, GenomicRanges 1.38.0, IRanges 2.20.0, MutationalPatterns 1.12.0, NMF 0.21.0, S4Vectors 0.24.0, TxDb.Hsapiens.UCSC.hg19.knownGene 3.2.2, XVector 0.26.0, bigmemory 4.5.33, biomaRt 2.42.0, cluster 2.1.0, doParallel 1.0.15, foreach 1.4.7, ggplot2 3.2.1, gridExtra 2.3, iterators 1.0.12, pkgmaker 0.27, registry 0.5-1, rngtools 1.4, rtracklayer 1.46.0
- Loaded via a namespace (and not attached): BiocFileCache 1.10.0, BiocManager 1.30.9, BiocParallel 1.20.0, BiocStyle 2.14.0, DBI 1.0.0, DelayedArray 0.12.0, GenomeInfoDbData 1.2.2, GenomicAlignments 1.22.0, MASS 7.3-51.4, Matrix 1.2-17, R6 2.4.0, RColorBrewer 1.1-2, RCurl 1.95-4.12, RSQLite 2.1.2, Rcpp 1.0.2, Rsamtools 2.2.0, SummarizedExperiment 1.16.0, VariantAnnotation 1.32.0, XML 3.98-1.20, askpass 1.1, assertthat 0.2.1, backports 1.1.5, bibtex 0.4.2, bigmemory.sri 0.1.3, bit 1.1-14, bit64 0.9-7, bitops 1.0-6, blob 1.2.0, codetools 0.2-16, colorspace 1.4-1, compiler 3.6.1, cowplot 1.0.0, crayon 1.3.4, curl 4.2, dbplyr 1.4.2, digest 0.6.22, dplyr 0.8.3, evaluate 0.14, ggdendro 0.1-20, glue 1.3.1, grid 3.6.1, gridBase 0.4-7, gtable 0.3.0, hms 0.5.1, htmltools 0.4.0, httr 1.4.1, knitr 1.25, labeling 0.3, lattice 0.20-38, lazyeval 0.2.2, magrittr 1.5, matrixStats 0.55.0, memoise 1.1.0, munsell 0.5.0, openssl 1.4.1, pillar 1.4.2, pkgconfig 2.0.3, plyr 1.8.4, pracma 2.2.5, prettyunits 1.0.2, progress 1.2.2, purrr 0.3.3, rappdirs 0.3.1, reshape2 1.4.3, rlang 0.4.1, rmarkdown 1.16, scales 1.0.0, stringi 1.4.3, stringr 1.4.0, tibble 2.1.3, tidyselect 0.2.5, tools 3.6.1, vctrs 0.2.0, withr 2.1.2, xfun 0.10, xtable 1.8-4, yaml 2.2.0, zeallot 0.1.0, zlibbioc 1.32.0