Introduction to *MutationalPatterns*

Francis Blokzijl Roel Janssen Ruben van Boxtel Edwin Cuppen

May 18, 2017

Contents

1 Introduction

Mutational processes leave characteristic footprints in genomic DNA. This package provides an easy-to-use toolset for the characterization and visualization 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 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 inferring the contribution of previously identified mutational signatures in a given sample. MutationalPatterns integrates with common R genomic analysis workflows and allows easy association with (publicly available) annotation data.

This package provides a comprehensive set of flexible functions for easy finding and plotting of such mutational patterns in base substitution catalogues.

2 Related packages

2.1 Comparison to SomaticSignatures

2.1.1 Similar functionality

SomaticSignatures provides functions for genomic context determination and *de novo* signature extraction using NMF decomposition of 96 trinucleotide count matrices. MutationalPatterns provides this functionality too, but the plotting is different, because MutationalPatterns offers the functionality to extract signatures *de novo* from a 192 feature matrix, with 96 trinucleotide X 2 transcriptional strands. This allows assessment of transcriptional strand bias of mutational signatures, which is important to characterize the underlying mutational mechanism.

2.1.2 Unique functionality

The following functions can be found in MutationalPatterns, but not in SomaticSignatures.

- `plot_contribution`: A function to determine the optimal contribution of previously identified signatures, e.g. cosmic cancer signatures, to reconstruct the mutational profile of just a single sample. This is useful for two reasons:
    1. for NMF you need many samples with distinct mutational profiles, as it relies on dimensionality reduction.
    2. In order to further characterize "known" mutational signatures and find the underlying mutational mechanisms, this function can be used to determine the contribution of these signatures in different samples, e.g. normal cells, or cells with defective DNA repair mechanisms etc.
- `plot_enrichment_depletion`: A plotting function and statistical test for enrichment or depletion of mutations in any (publicly available) annotated genomic region such as transcription factor binding site or "open chromatin". This is useful for the characterization of mutational mechanisms.
- `strand_bias_test`, `plot_strand_bias`: A statistical test and a plotting function for transcriptional strand bias in mutation catalogs.
- `plot_compare_profiles`: A plotting function to visualize the difference between two 96 mutational profiles and calculate RSS.

3 Data

To perform mutational pattern analyses you need your VCF datasets and the reference genome.

3.1 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)
```

3.2 Load example data

We provided a small example data set with this package. The data consists of somatic mutation catalogues of 9 normal human adult stem cells from 3 tissues (Blokzijl et al., 2016).

Load the MutationalPatterns package:

```
> library(MutationalPatterns)
```

Locate the example data:

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

Define corresponding sample names for the datasets:

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

This package uses GRanges to represent the variant calls. So, to work with data, we need to load it as such.

Load the example VCF files into a GRangesList:

```
> vcfs <- read_vcfs_as_granges(vcf_files, sample_names, ref_genome)
> summary(vcfs)
```

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

Store relevant metadata on the samples, such as tissue type, in a character vector:

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

4 Mutation characteristics

Now we have loaded the VCF files and the corresponding reference genome, we can start our search for characteristic mutational footprints.

4.1 Base substitution types

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

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

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

We can retrieve base substitutions from the VCF 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), this functions returns the G:C>T:A mutation as a C>A mutation:

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

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

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

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

```
 chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1 
"TCG" "AAA" "CGG" "CTT" "GTC" "ATT" "ACG" "CGA" "TTT" "TCC" "ACG" "CCT" 
```

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

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

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

```
> head(type_context$context, 12)
```

```
 chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1  chr1 
"TCG" "TTT" "CCG" "CTT" "GTC" "ATT" "ACG" "TCG" "TTT" "TCC" "ACG" "CCT" 
```

Using `mut_type_occurrences`, you can count mutation type occurrences for the loaded samples in a list of VCF objects, which can be used to plot the mutation spectrum.

```
> 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      80  20 267  30  84  19        147       120
colon2      77  22 279  27  81  14        178       101
colon3      92  29 271  33  58  17        165       106
intestine1  71  30 258  49  69  23        140       118
intestine2  57  25 266  36  93  23        155       111
intestine3  56  20 285  33  80  26        176       109
liver1      75  56 152  49 118  50         32       120
liver2      94  63 147  47 126  23         23       124
liver3     105  66 150  47  94  38         26       124
```

4.2 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 n indicates the total number of mutations in the set.

```
> p1 = plot_spectrum(type_occurrences)
```

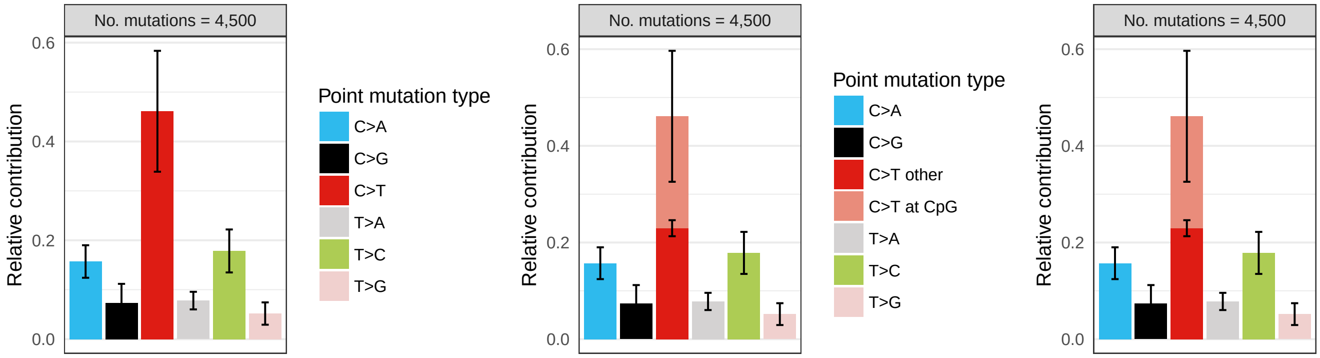
Plot the mutation spectrum with distinction between C>T at CpG sites:

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

Plot spectrum without legend:

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

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



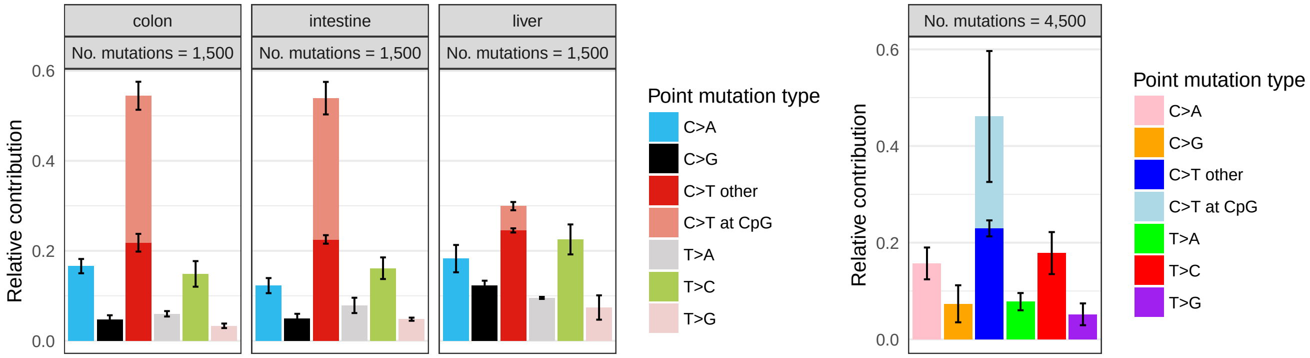
You can facet the per sample group, e.g. plot spectrum for each tissue separately:

```
> 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))
```



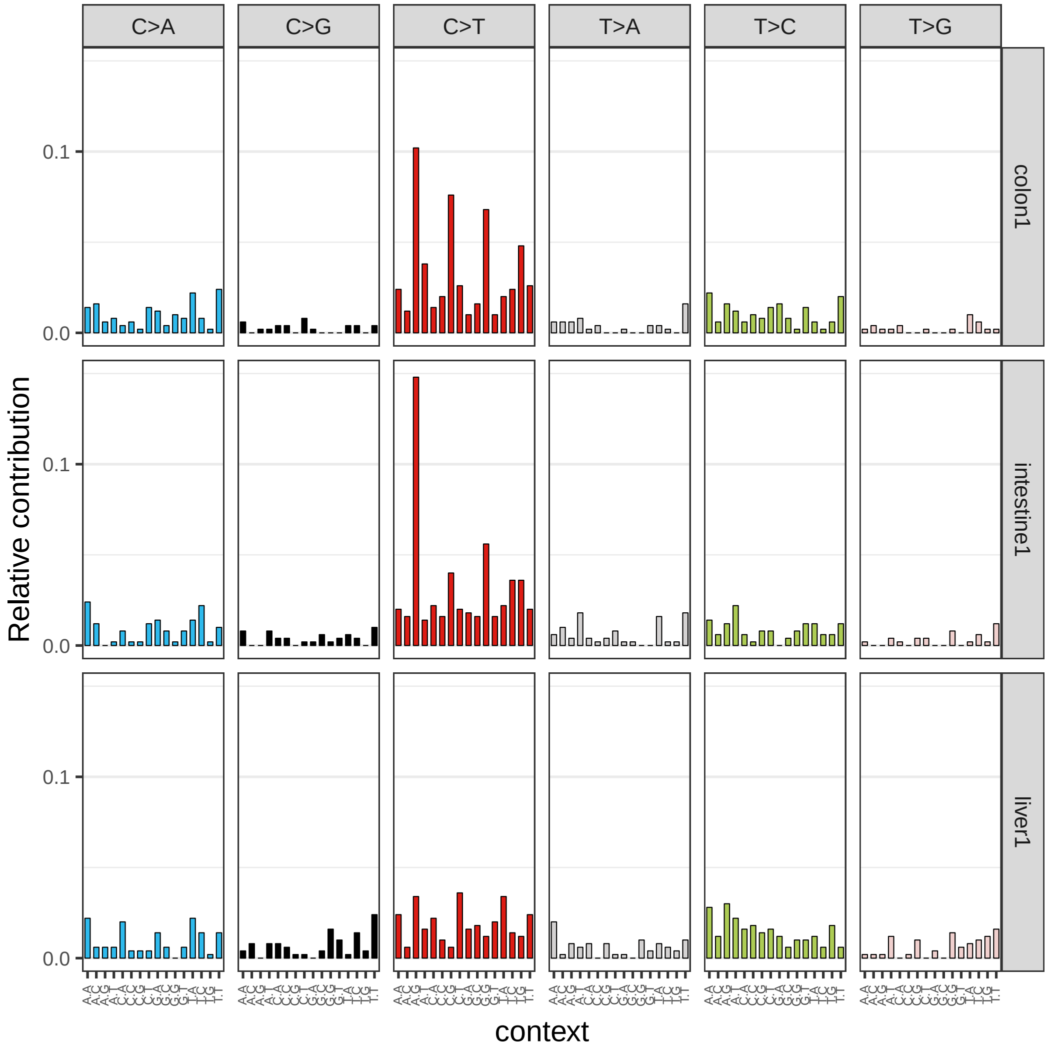
4.3 96 Mutation profile

Make 96 trinucleodide mutation count matrix:

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

Plot 96 profile of three samples:

```
> plot_96_profile(mut_mat[,c(1,4,7)])
```



5 Mutational signatures

5.1 De novo mutational signature extraction using NMF

A critical parameter in NMF is the factorization rank, which is the number of mutational signatures. 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."

We can use the NMF package to determine which rank we should use to extract signatures using `extract_signatures`:

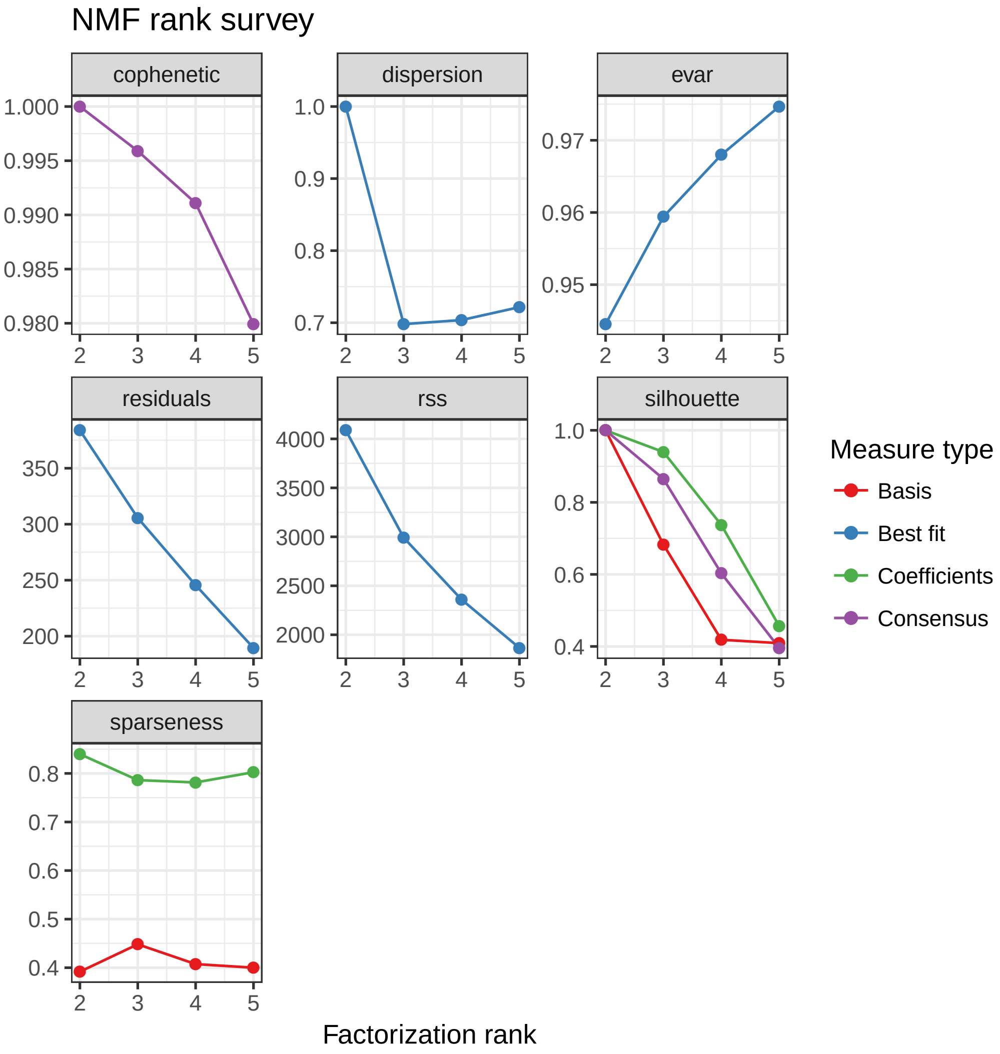
Add a small psuedocount to avoid a 0 in the matrix:

```
> mut_mat = mut_mat + 0.0001
```

Use the NMF package to generate an estimate plot.

```
> library("NMF")
> estimate = nmf(mut_mat, rank=2:5, method="brunet", nrun=100, seed=123456)
```

```
> plot(estimate)
```



Extract e.g. 2 mutational signatures from the mutation count matrix:

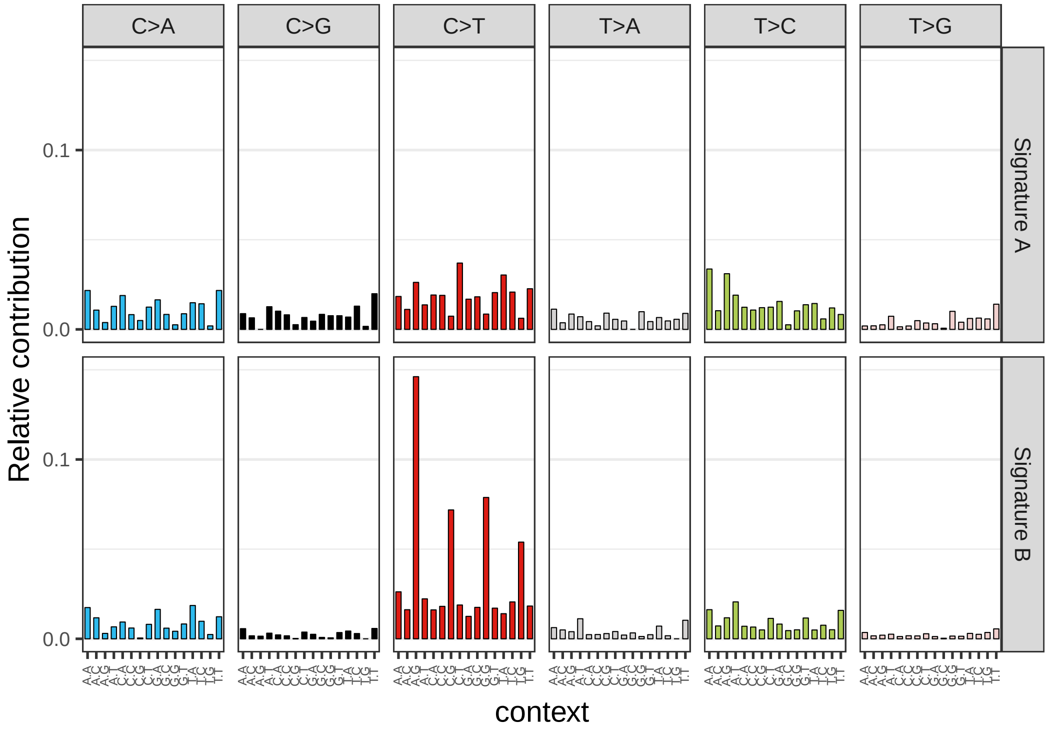
```
> nmf_res <- extract_signatures(mut_mat, rank = 2)
```

Provide column names for the signatures:

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

Plot the 96-profile of the signatures:

```
> plot_96_profile(nmf_res$signatures)
```

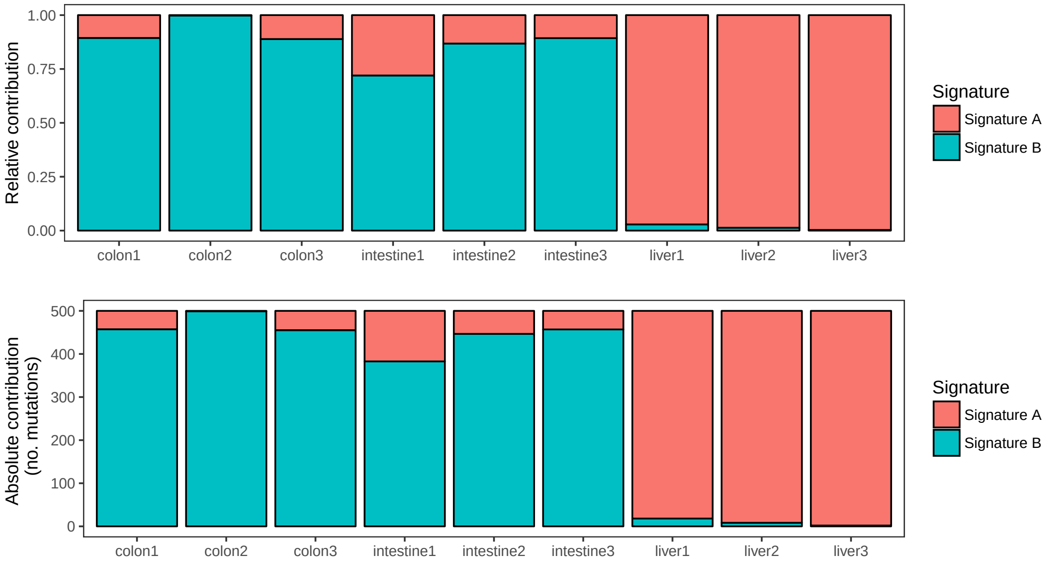


Plot the contribution of the signatures:

```
> rownames(nmf_res$contribution) <- c("Signature A", "Signature B")
> pc1 <- plot_contribution(nmf_res$contribution, nmf_res$signature, mode = "relative")
```

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

```
> pc2 <- plot_contribution(nmf_res$contribution, nmf_res$signature, mode = "absolute")
> grid.arrange(pc1, pc2)
```

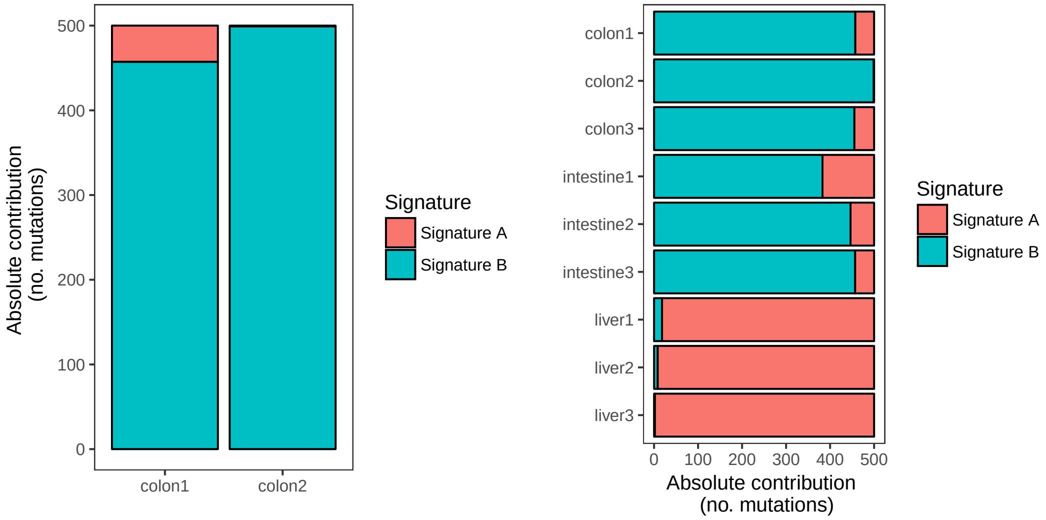


Plot contribution of signatures for subset of samples with index parameter:

```
> pc3 <- plot_contribution(nmf_res$contribution, nmf_res$signature,
+                          mode = "absolute", index = c(1,2))
```

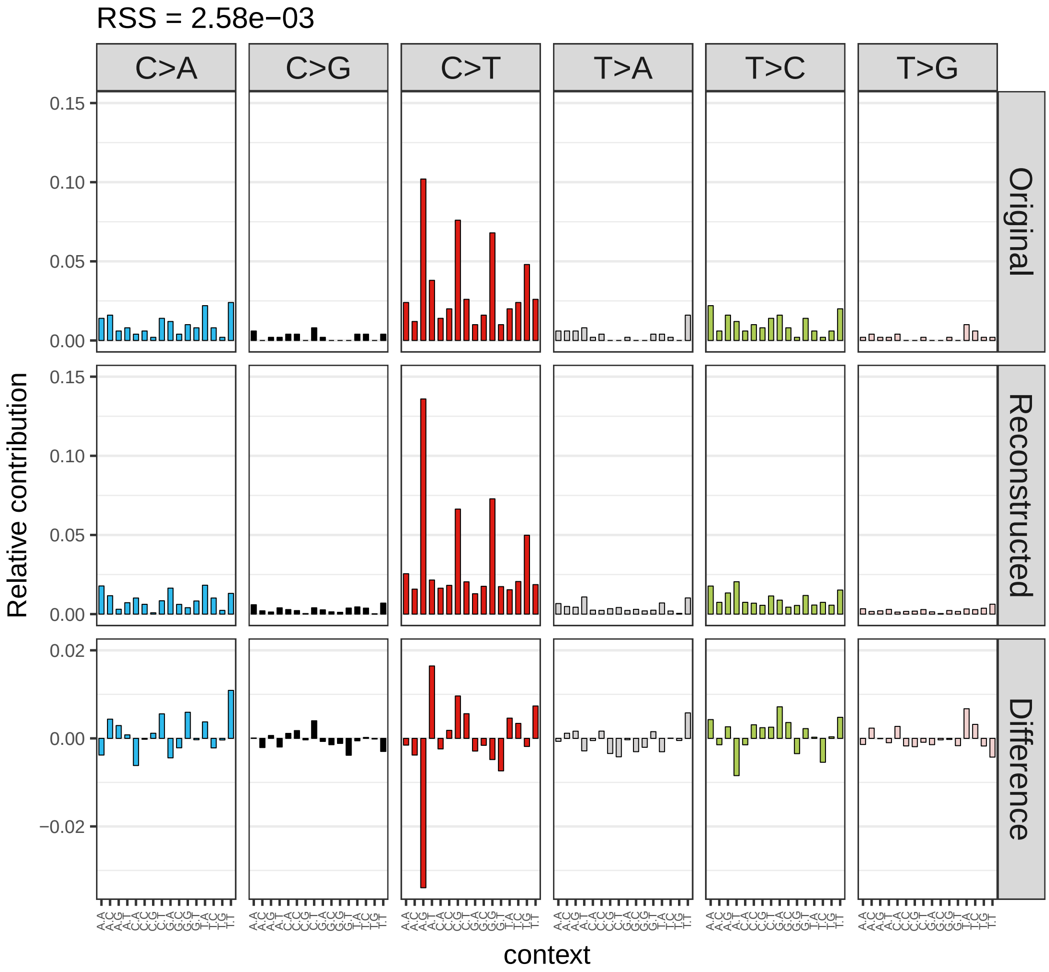
Flip X and Y coordinates:

```
> pc4 <- plot_contribution(nmf_res$contribution, nmf_res$signature,
+                          mode = "absolute", coord_flip = TRUE)
> grid.arrange(pc3, pc4, ncol=2)
```



Compare reconstructed mutation profile with original mutation profile:

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



5.2 Fit 96 mutation profiles to known signatures

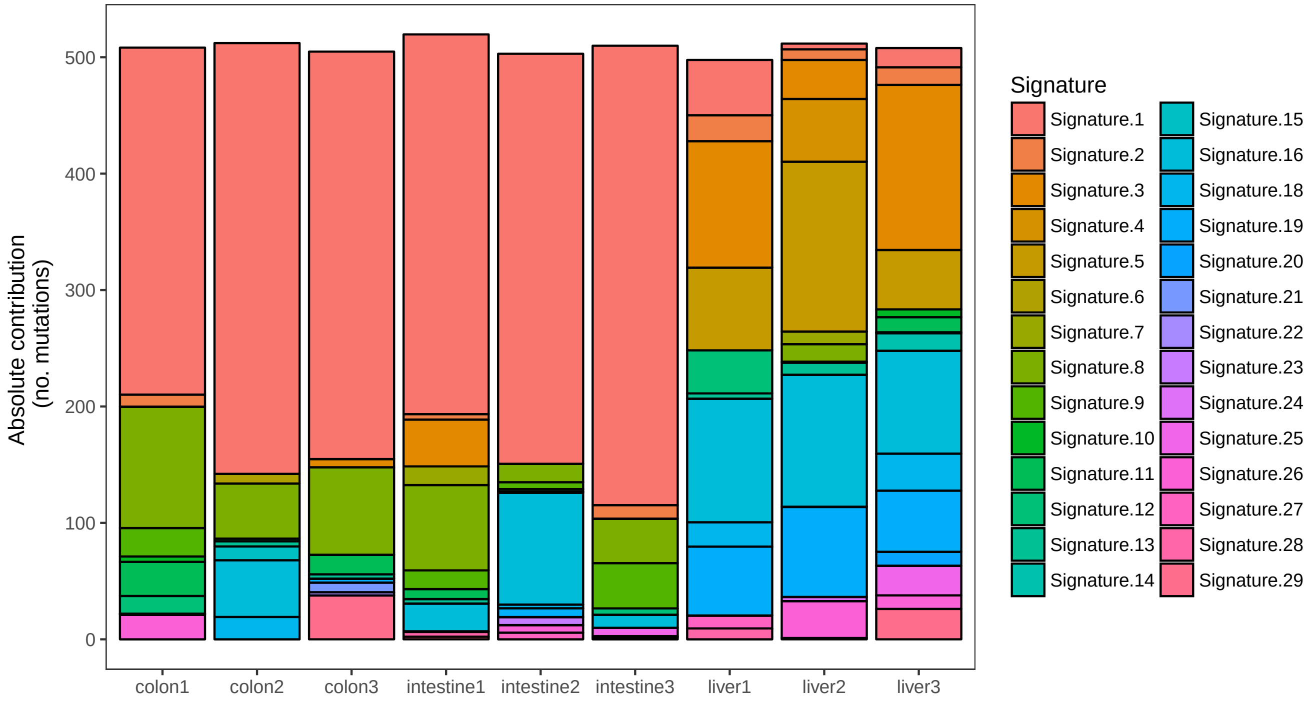
Download signatures from pan-cancer study (Alexandrov et al., 2013):

```
> sp_url <- paste("http://cancer.sanger.ac.uk/cancergenome/assets/",
+                 "signatures_probabilities.txt", sep = "")
> cancer_signatures = read.table(sp_url, sep = "\t", header = TRUE)
> # Reorder (to make the order of the trinucleotide changes the same)
> cancer_signatures = cancer_signatures[order(cancer_signatures[,1]),]
> # Only signatures in matrix
> cancer_signatures = as.matrix(cancer_signatures[,4:33])
```

Fit mutation matrix to cancer signatures. This function finds the optimal linear combination of mutation signatures that most closely reconstructs the mutation matrix by solving a non-negative least-squares constraints problem.

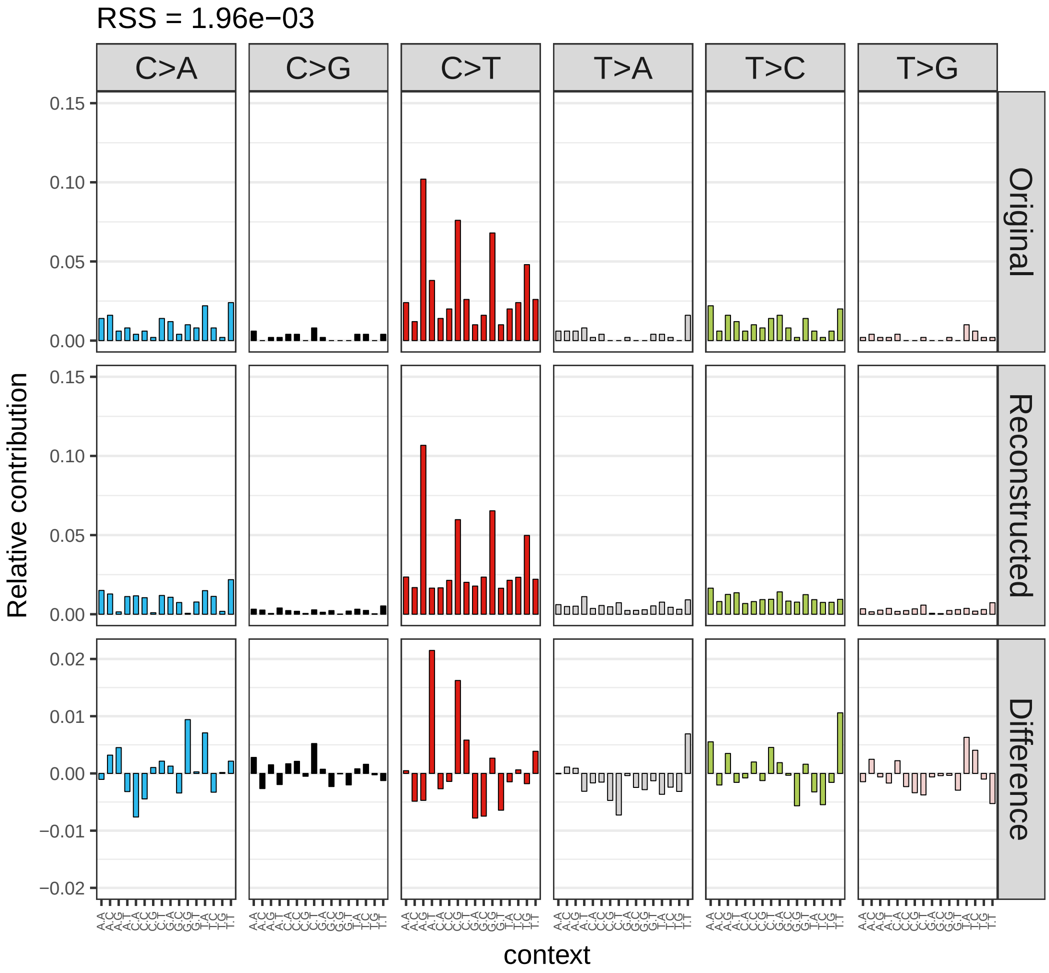
```
> fit_res <- fit_to_signatures(mut_mat, cancer_signatures)
```

```
> # select signatures with some contribution
> select <- which(rowSums(fit_res$contribution) > 0)
> # plot contribution
> plot_contribution ( fit_res$contribution[select,],
+                     cancer_signatures[,select],
+                     coord_flip = FALSE,
+                     mode = "absolute" )
```



Compare reconstructed mutation profile of sample 1 using cancer signatures with original profile:

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



6 Transcriptional strand bias

6.1 Strand bias analysis

For the mutations within genes it can be determined whether the mutation is on the transcribed or non-transcribed strand, which is interesting to study 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.

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)
```

Get transcriptional strand information for all positions in the first VCF object. Function returns "-" for positions outside gene bodies, "U" for untranscribed/sense/coding strand, "T" for transcribed/anti-sense/non-coding strand.

```
> strand = strand_from_vcf(vcfs[[1]], genes_hg19)
> head(strand, 10)
```

```
 [1] "T" "-" "-" "-" "-" "-" "-" "-" "T" "-"
```

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)
> head(mut_mat_s, 10)
```

```
      colon1 colon2 colon3 intestine1 intestine2 intestine3 liver1 liver2 liver3
ACA-u      1      0      3          0          0          1      0      0      2
ACA-t      1      1      1          3          0          1      3      4      1
ACC-u      2      1      0          2          1          0      0      2      1
ACC-t      1      1      0          1          1          2      0      1      0
ACG-u      1      0      0          0          0          0      1      0      0
ACG-t      0      1      1          0          0          0      0      1      1
ACT-u      1      2      0          1          0          0      0      0      0
ACT-t      0      0      2          0          0          1      0      2      1
CCA-u      0      1      4          1          1          1      2      2      0
CCA-t      1      0      1          1          1          0      1      3      1
```

Perform strand bias analysis:

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

```
   group type strand no_mutations relative_contribution
1  colon  C>A      T           50            0.09803922
4  colon  C>A      U           36            0.07058824
7  colon  C>G      T           17            0.03333333
10 colon  C>G      U           15            0.02941176
13 colon  C>T      T          123            0.24117647
16 colon  C>T      U          149            0.29215686
```

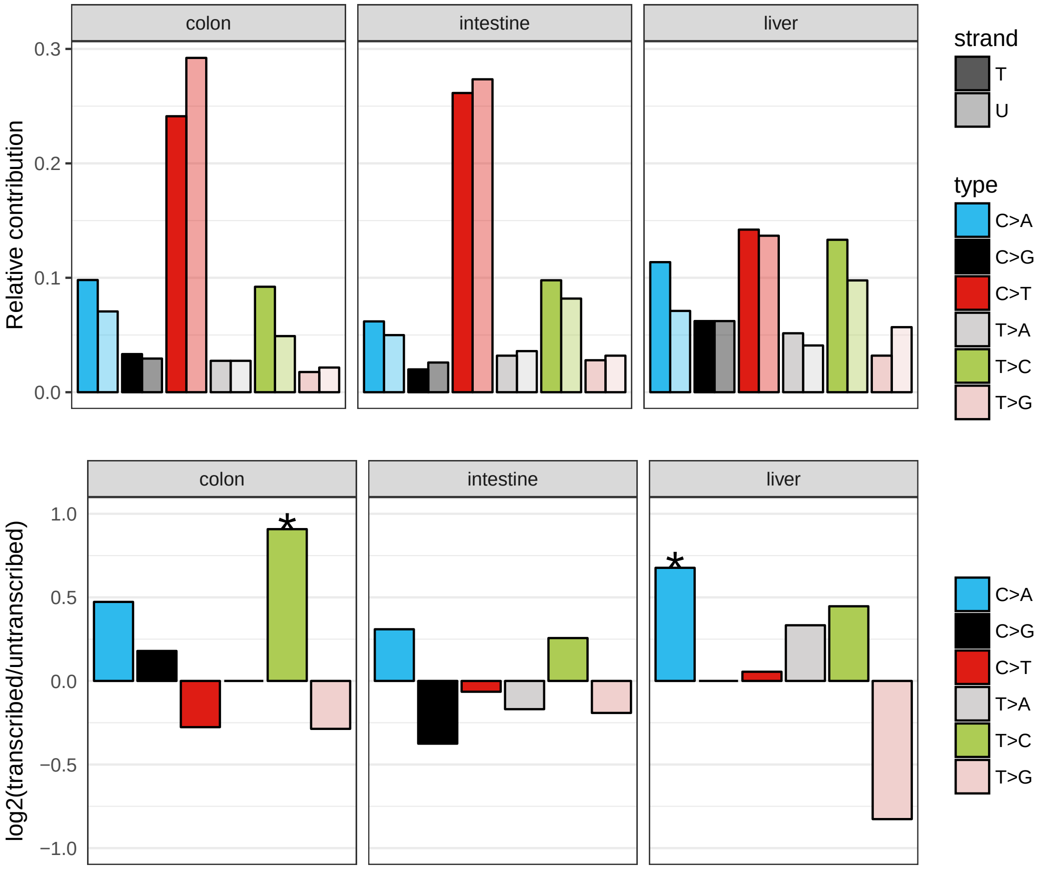
```
> ps1 <- plot_strand(strand_counts, mode = "relative")
```

Perform poisson test for strand asymmetry significance testing:

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

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

```
> ps2 <- plot_strand_bias(strand_bias)
> grid.arrange(ps1, ps2)
```



6.2 Extract signatures with strand bias

Extract 2 signatures with strand bias:

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

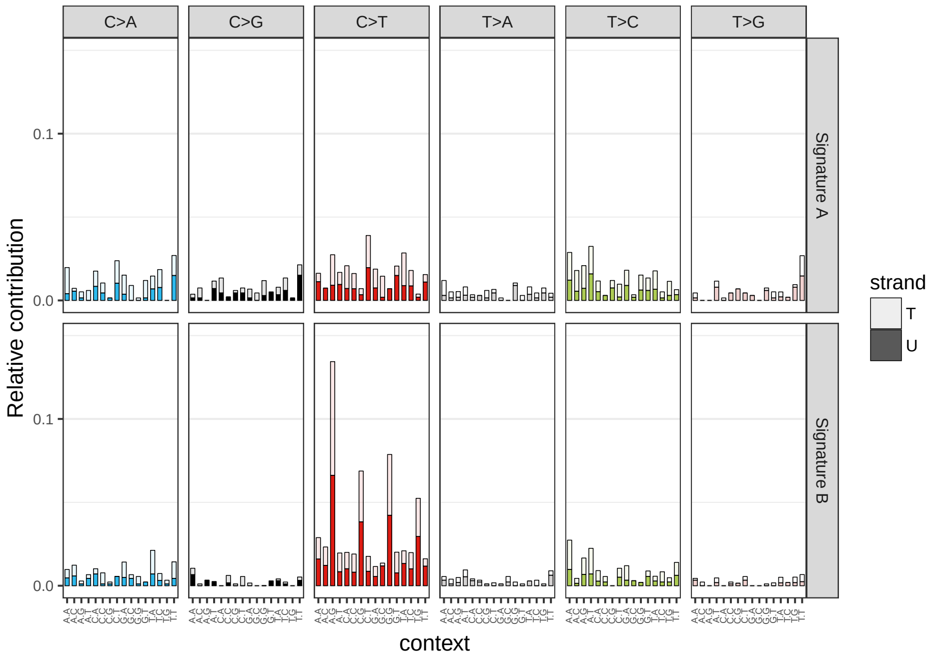
Plot signatures with 192 features:

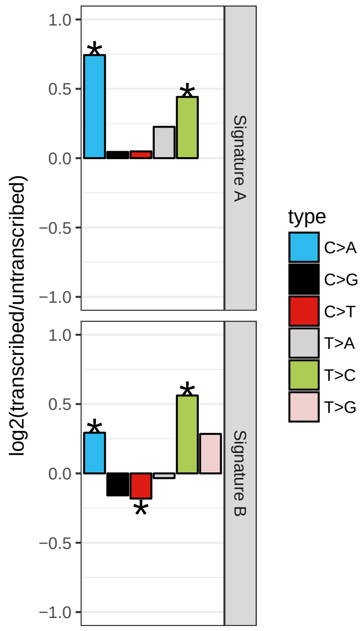
```
> a <- plot_192_profile(nmf_res_strand$signatures)
```

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

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

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





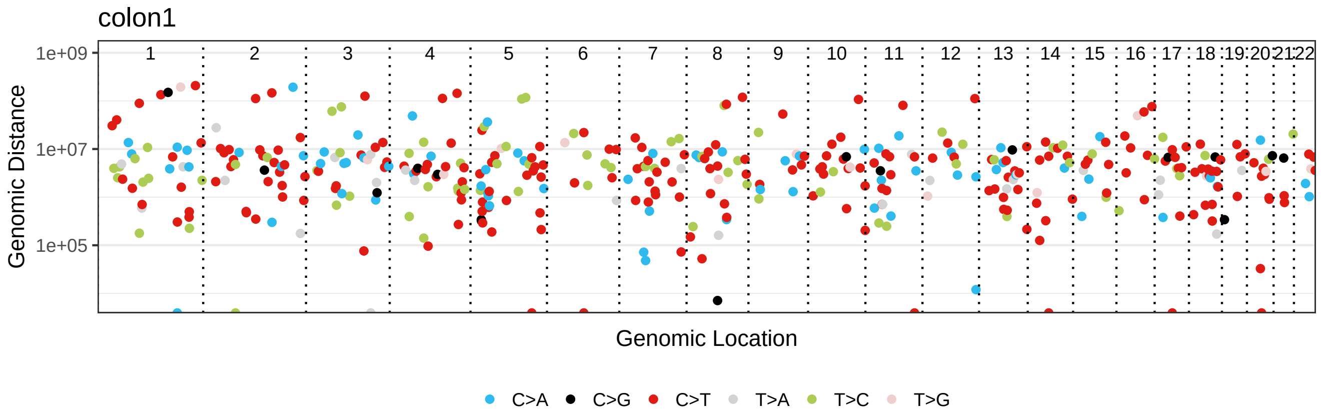
7 Genomic distribution

7.1 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.

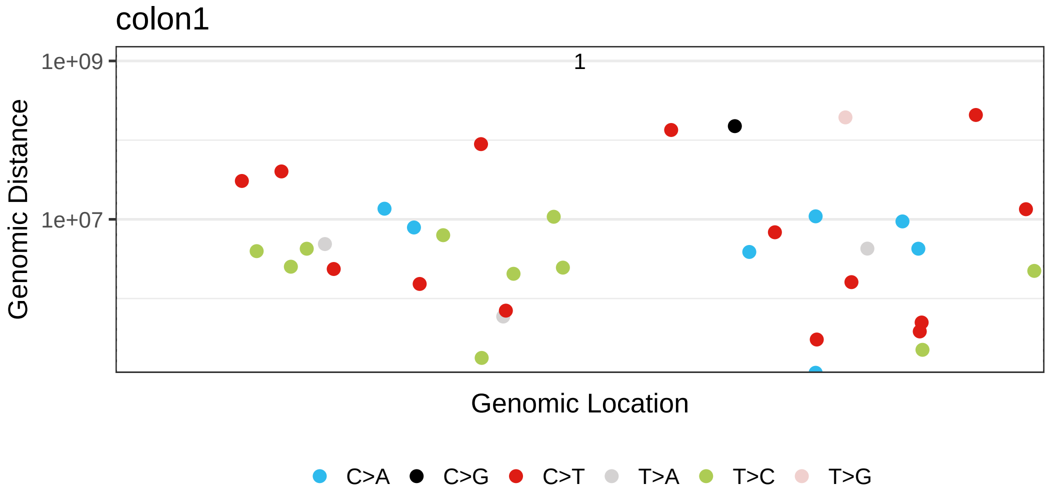
Make rainfall plot of sample 1 over all autosomal chromosomes

```
> # 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 )
```



Make rainfall plot of the first sample over chromosome 1:

```
> chromosomes <- seqnames(get(ref_genome))[1]
> plot_rainfall ( vcfs[[1]], title = names(vcfs[1]),
+                 chromosomes = chromosomes[1], cex = 2, ylim = 1e+09 )
```



7.2 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 publically available genomic annotation data, like in the example below.

7.2.1 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")
```

Don't forget to use the same chromosome naming convention consistently:

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

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

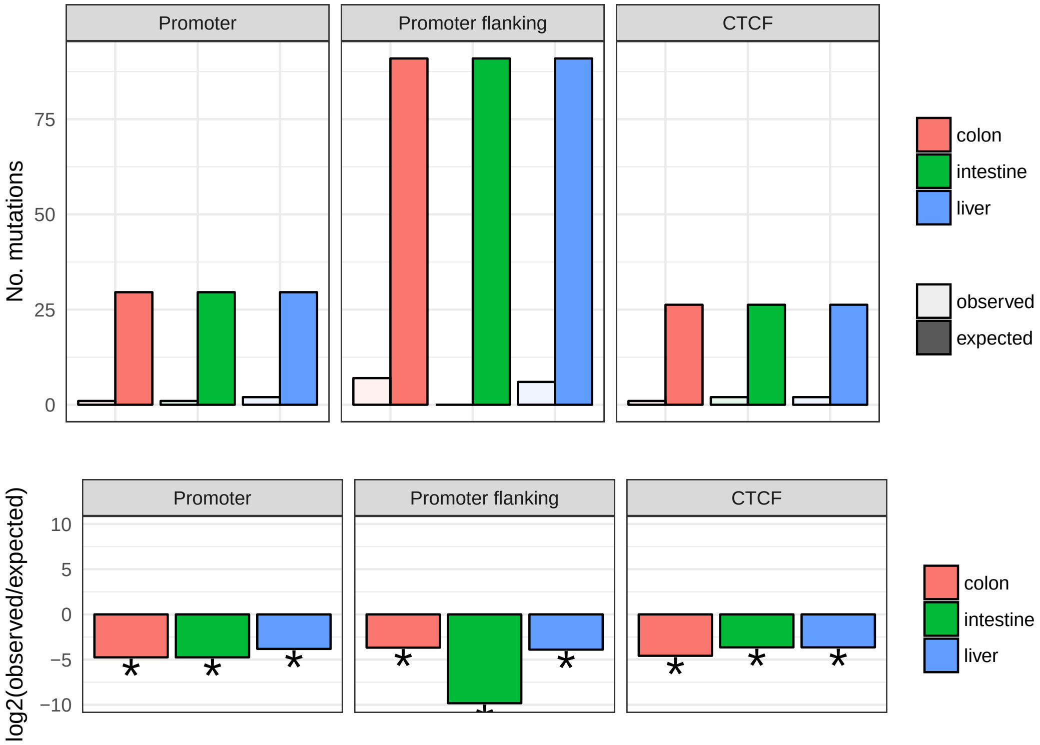
```
         by            region n_muts surveyed_length surveyed_region_length observed
1     colon          Promoter   1500       727070334               14327310        1
2 intestine          Promoter   1500       727070334               14327310        1
3     liver          Promoter   1500       727070334               14327310        2
4     colon Promoter flanking   1500       727070334               44087613        7
5 intestine Promoter flanking   1500       727070334               44087613        0
6     liver Promoter flanking   1500       727070334               44087613        6
          prob expected    effect         pval significant
1 2.063074e-06 29.55830 depletion 4.447430e-12           *
2 2.063074e-06 29.55830 depletion 4.447430e-12           *
3 2.063074e-06 29.55830 depletion 6.802585e-11           *
4 2.063074e-06 90.95601 depletion 3.483525e-30           *
5 2.063074e-06 90.95601 depletion 3.149664e-40           *
6 2.063074e-06 90.95601 depletion 2.649785e-31           *
```

```
> ## Or without specifying the 'by' parameter.
> distr_test2 <- enrichment_depletion_test(distr)
> head(distr_test2)
```

```
          by   region n_muts surveyed_length surveyed_region_length observed         prob
1     colon1 Promoter    500       242356778                4775770        1 2.063074e-06
2     colon2 Promoter    500       242356778                4775770        0 2.063074e-06
3     colon3 Promoter    500       242356778                4775770        0 2.063074e-06
4 intestine1 Promoter    500       242356778                4775770        0 2.063074e-06
5 intestine2 Promoter    500       242356778                4775770        0 2.063074e-06
6 intestine3 Promoter    500       242356778                4775770        1 2.063074e-06
```

```
  expected    effect         pval significant
1 9.852768 depletion 5.708662e-04           *
2 9.852768 depletion 5.260088e-05           *
3 9.852768 depletion 5.260088e-05           *
4 9.852768 depletion 5.260088e-05           *
5 9.852768 depletion 5.260088e-05           *
6 9.852768 depletion 5.708662e-04           *

> plot_enrichment_depletion(distr_test)
TableGrob (2 x 1) "arrange": 2 grobs
  z     cells    name           grob
1 1 (1-1,1-1) arrange gtable[layout]
2 2 (2-2,1-1) arrange gtable[layout]
```



References

Alexandrov, L. B., Nik-Zainal, S., Wedge, D. C., Aparicio, S. A. J. R., Behjati, S., Biankin, A. V., . . . Stratton, M. R. (2013, Aug 22). Signatures of mutational processes in human cancer. *Nature*, *500*(7463), 415–421. Retrieved from http://dx.doi.org/10.1038/nature12477 (Article)

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

8 Session Information

- R version 3.4.0 (2017-04-21), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 16.04.2 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.5-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.5-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.38.0, BSgenome 1.44.0, BSgenome.Hsapiens.UCSC.hg19 1.4.0, Biobase 2.36.2, BiocGenerics 0.22.0, Biostrings 2.44.0, GenomeInfoDb 1.12.0, GenomicFeatures 1.28.0, GenomicRanges 1.28.2, IRanges 2.10.1, MutationalPatterns 1.2.1, NMF 0.20.6, S4Vectors 0.14.1, TxDb.Hsapiens.UCSC.hg19.knownGene 3.2.2, XVector 0.16.0, bigmemory 4.5.19, bigmemory.sri 0.1.3, biomaRt 2.32.0, cluster 2.0.6, doParallel 1.0.10, foreach 1.4.3, ggplot2 2.2.1, gridExtra 2.2.1, iterators 1.0.8, pkgmaker 0.22, registry 0.3, rngtools 1.2.4, rtracklayer 1.36.2
- Loaded via a namespace (and not attached): BiocInstaller 1.26.0, BiocParallel 1.10.1, BiocStyle 2.4.0, DBI 0.6-1, DelayedArray 0.2.3, GenomeInfoDbData 0.99.0, GenomicAlignments 1.12.1, Matrix 1.2-10, RColorBrewer 1.1-2, RCurl 1.95-4.8, RSQLite 1.1-2, Rcpp 0.12.10, Rsamtools 1.28.0, SummarizedExperiment 1.6.1, VariantAnnotation 1.22.0, XML 3.98-1.7, backports 1.0.5, bitops 1.0-6, codetools 0.2-15, colorspace 1.3-2, compiler 3.4.0, digest 0.6.12, evaluate 0.10, grid 3.4.0, gridBase 0.4-7, gtable 0.2.0, htmltools 0.3.6, knitr 1.15.1, labeling 0.3, lattice 0.20-35, lazyeval 0.2.0, magrittr 1.5, matrixStats 0.52.2, memoise 1.1.0, munsell 0.4.3, plyr 1.8.4, pracma 2.0.4, quadprog 1.5-5, reshape2 1.4.2, rlang 0.1, rmarkdown 1.5, rprojroot 1.2, scales 0.4.1, stringi 1.1.5, stringr 1.2.0, tibble 1.3.1, tools 3.4.0, xtable 1.8-2, yaml 2.1.14, zlibbioc 1.22.0