Reference Manual: Sasquatch R-Tool

Version:	0.1
Date:	June 16, 2017
Authors:	Ron Schwessinger, Maria Suciu, Simon J McGowan, Jelena M Telenius, Douglas R. Higgs & Jim R. Hughes
Groups:	Genome Biology and Computational Biology Research Group, WIMM, Oxford
Contact:	Ron Schwessinger, ron.schwessinger@msdtc.ox.ac.uk CBRG, managers@molbiol.ox.ac.uk Jim Hughes, jim.hughes@imm.ox.ac.uk
License:	GPLv3
Requires:	ggplot2, RColorBrewer
Recommends:	Biostrings, TFBSTools, pbapply

Contents

CalcSFR

Description

Calculate the Shoulder-to-Footprint-Ratio of a given footprint profile and estimated shoulder positions and ranges.

Usage

```
    CalcSFR(profile,
            upstream.shoulder.position,
            upstream.shoulder.position,
            upstream.shoulder.range,
            upstream.shoulder.range)
```

Arguments

profile	input cut profile (should be smoothed)
us.mid	center bp position of the upstream shoulder (as estimated from *SobelBorders*)
ds.mid	center bp position of the downstream shoulder
range.us	range in bp of the upstream shoulder (as estimated from *SobelBorders*)
range.ds	range in bp of the downstream shoulder

Value

Returns the SFR as single numeric value.

Examples

```
frag.type <- "DNase"

data.dir <- "./my_sasq_database/human/DNase/"

tissue <- "blood_tissue"

kmer <- "CACGTG"

# get the average profile for a single kmer
profile<- GetFootprint(
  kmer=kmer,
  tissue=tissue,
  data.dir=data.dir,
  frag.type=frag.type,
  smooth=TRUE
  )
```

```
# estimate the shoulders from the profile
sh <- SobelBorders(profile, kl=nchar(kmer))

#calculate the SFR
CalcSFR(profile, sh$us, sh$ds, sh$range.us, sh$range.ds)
```

CompareSequences

Description

Wrapper function to compare two input sequences on a kmer basis. Both sequence are split up and compared pairwise, calculating the damage. Total damage is calculated either as summed up or as highest pairwise damage, as chosen. Overlay plots for none, all or only the highest scoring pair are created as specified.
(If preload = TRUE and if vocab.flag = FALSE, preload.profiles has to be submitted. If vocab.flag = TRUE with preload = TRUE then preload.vocab has to be submitted. If preload =TRUE and plots shoudl be generated, preload.profiles have to be provided.)

Usage

```
CompareSequences(sequence1,
                 sequence2,
                 kl,
                 damage.mode="exhaustive",
                 tissue,
                 data.dir,
                 pnorm.tag,
                 vocab.flag=FALSE,
                 vocab.file=paste0(data.dir,"/",tissue,
                           "/vocabulary_",tissue,".txt"),
                 frag.type,
                 plots="highest",
                 smooth=TRUE,
                 ylim=c(0,0.01),
                 xlim=c(-125,125),
                 plot.shoulders=FALSE,
                 preload,
                 preload.vocab="",
                 preload.profiles="")
```

Arguments

sequence1	input sequence 1, character string (usually length of kmer or a window over a variant)
sequence2	input sequence 2, character string, required to have same length as sequence 1
kl	length of the kmer windows, to split up the sequences
damage.mode	{"exhaustive", "local"} mode to calculate the total damage for the comparison. **"exhaustive"** sum over all pair-wise comparison. **"local"** extract the highest pair-wise damage score. [default="exhaustive"]
tissue	name of tissue directory of interest
data.dir	path to directory containing preprocessed data
pnorm.tag	tag indicating which background propensities were used for normalisation
vocab.flag	{FALSE, TRUE} flag if a precalculated vocabulary file is present and should be used [default=FALSE] Speeds up the analysis but precalculation is time consuming, therefore has to be done manually after initial processing of new tissue data. Recommended if tissue is frequently used for analysis.
vocab.file	path to precalculated vocabulary file [default=paste0(data.dir,"",tissue,"vocabulary_",tissue,".txt")]
frag.type	{"DNase", "ATAC"} fragmentation type.
plots	{FALSE, "highest", "all"} indicate if and what overlay plots to retrieve [default="highest"] **FALSE** no plots, **highest** only the highest scoring pair, **"all"** plots for all pairs
smooth	{FALSE, TRUE} smooth the profile [default=TRUE]
ylim	y-limit for plots [default c(0,0.01)]
xlim	x-limit for plots [default c(-125,125)]
plot.shoulders	{FALSE, TRUE} print the estimated shoulders with the overlay plots [default=FALSE]
preload	flag [FALSE/TRUE] if to use preloaded average profiles [default=FALSE]
preload.vocab	preloaded vocabulary data frame from PreLoadVocab()
preload.profiles	two dataframe list with the preloaded profiles from PreLoadProfiles()

Value

Returns a list containing two data frames and a plot or list of plots, depending on how the plotting was specified.
...$summary: a data frame listing the reference and variant sequence with highest scoring kmer pair
...$df: data frame containing the pair-wise results per row (kmer and SFR from reference and variant and the damage)
...$plot: Overlay plot(s), either single ggplot2 plot or list of plots depending on mode selected for "plots"

Examples

```
frag.type <- "DNase"
data.dir <- "./my_sasq_database/human/DNase/"
tissue <- "blood_tissue"
pnorm.tag <- "h_ery_1"

comp <- CompareSequences(
  sequence1="CAGTTTCATGAGG",
  sequence2="CAGTTTTATGAGG",
  kl=7,
  data.dir=data.dir,
```

```
  pnorm.tag=pnorm.tag,
  damage.mode="exhaustive",
  tissue=tissue,
  vocab.flag=TRUE,
  frag.type="DNase",
  plots="highest"
  )

#With preload
vocabulary <- PreLoadVocab(data.dir, tissue)

comp <- CompareSequences(
  sequence1="CAGTTTCATGAGG",
  sequence2="CAGTTTTATGAGG",
  kl=7,
  data.dir=data.dir,
  pnorm.tag=pnorm.tag,
  damage.mode="exhaustive",
  tissue=tissue,
  vocab.flag=TRUE,
  frag.type="DNase",
  plots="highest",
  preload=TRUE,
  preload.vocab=vocabulary)
```

DecodeKmer

Description

Takes a kmer and dissects all ambivalent FASTA characters to create a character vector of all matching kmers.

Usage

```
DecodeKmer(kmer)
```

Arguments

kmer	character string, allowing all FASTA standard and ambivalent characters

Value

Returns a vector containing of all matching definite kmers as character strings.

Examples

```
DecodeKmer("WGATAA")
```

DissectSequence

Description

Split an input sequence into kmer windows.

Usage

```
DissectSequence(sequence, kl, list=FALSE)
```

Arguments

sequence	character string, allowing all FASTA standard and ambivalent characters
kl	length in bp to split the sequence into
list	{FALSE, TRUE} select if output should be a list or a vector [default=FALSE]

Value

Returns a vector or list of all kmers.

Examples

```
DissectList("AGGGATACGTAGACGGTGTAA", kl=7, list=FALSE)
```

GetCount

Description

Wrapper function to get the count of the k-mer in DHS in the tissue of interest.

Usage

```
GetCount(kmer, tissue, data.dir, pnorm.tag, frag.type)
```

Arguments

kmer	input kmer, FASTA characters string 5 - 7 bp
tissue	name of tissue directory of interest
data.dir	path to directory containing preprocessed data
pnorm.tag	tag indicating which background propensities were used for normalisation
frag.type	{"DNase", "ATAC"} fragmentation type.

Value

Returns the count as single numeric value.

Examples

```
frag.type <- "DNase"
data.dir <- "./my_sasq_database/human/DNase/"
tissue <- "blood_tissue"
pnorm.tag <- "h_ery_1"

GetCount(kmer="CACGTG", data.dir=data.dir, tissue=tissue, pnorm.tag=pnorm.tag,
         frag.type=frag.type)
```

GetFootprint

Description

Wrapper to retrieve the merged & smoothed profile of kmer.

Usage

```
GetFootprint(kmer,
             tissue,
             data.dir,
             pnorm.tag,
             frag.type,
             smooth=TRUE,
             smooth.bandwidth=5,
             preload=FALSE,
             preload.profiles="")
```

Arguments

kmer	input kmer, FASTA characters string 5 - 7 bp
tissue	name of tissue directory of interest
data.dir	path to directory containing preprocessed data
pnorm.tag	tag indicating which background propensities were used for normalisation
frag.type	{"DNase", "ATAC"} fragmentation type.
smooth	{FALSE, TRUE} smooth the profile [default=TRUE]
smooth.bandwidth	bandwidth to smooth in bp [default=5]
preload	flag: [FALSE/TRUE] if to use preloaded average profiles [default=FALSE]
preload.profiles	two dataframe list with the preloaded profiles from PreLoadProfiles()

Value

Returns list containing the profile and counts. ...$profile, ...$count.

Examples

```
frag.type <- "DNase"
data.dir <- "./my_sasq_database/human/DNase/"
tissue <- "blood_tissue"
pnorm.tag <- "h_ery_1"

fp <- GetFootprint(kmer="CACGTG",
```

```
                    tissue=tissue,
                    pnorm.tag=pnorm.tag,
                    data.dir=data.dir,
                    frag.type=frag.type,
                    smooth=TRUE)

# With preloading
profiles <- PreLoadKmerProfiles(7, data.dir, tissue, pnorm.tag)
fp <- GetFootprint(kmer="CACGTG",
                    tissue=tissue,
                    data.dir=data.dir,
                    pnorm.tag=pnorm.tag,
                    frag.type=frag.type,
                    smooth=TRUE,
                    preload=TRUE,
                    preload.profiles=profiles)
```

GetFootprintStrand

Description

Wrapper to retrieve strand-specific (smoothed) profiles of a kmer from tissue or background

Usage

```
GetFootprintStrand(kmer,
                    tissue,
                    data.dir,
                    pnorm.tag,
                    frag.type,
                    smooth=TRUE,
                    smooth.bandwidth=5,
                    background.flag=FALSE)
```

Arguments

kmer	input kmer, FASTA characters string 5 - 7 bp
tissue	name of tissue directory of interest
data.dir	path to directory containing preprocessed data
pnorm.tag	tag indicating which background propensities were used for normalisation
frag.type	{"DNase", "ATAC"} fragmentation type.
smooth	{FALSE, TRUE} smooth the profile [default=TRUE]
smooth.bandwidth	bandwidth to smooth in bp [default=5]
background.flag	select if to retrieve the profile from the genome-wide background [default=FALSE]

Value

Returns list with strand-specific profiles and counts. ...$profile.plus, ...$profile.minus, ...$count.plus, ...$count.minus

Examples

```
frag.type <- "DNase"
data.dir <- "./my_sasq_database/human/DNase/"
tissue <- "blood_tissue"
pnorm.tag <- "h_ery_1"

GetFootprintStrand(kmer="CACGTG",
                   tissue=tissue,
                   data.dir=data.dir,
                   pnorm.tag=pnorm.tag,
                   frag.type=frag.type,
                   smooth=TRUE,
                   background.flag=FALSE)
```

GetPossibleMutations

Description

Take a sequence input and split into kmers of length $kl * 2 - 1$ (e.g. for kl 7 always takes the 13 surrounding bases) and separate into reference and variance. Take the parsed position as index for the first base to mutate. List all possible mutations filling the kl'th position in the variant column with all possible substituions depneding on the reference base. First position of the seuqence to be mutated is therefore the kl'th positoin of the sequence string.

Usage

```
GetPossibleMutations(sequence, kl=7, chr=".", position=1)
```

Arguments

sequence	character string, allowing all FASTA standard and ambivalent characters
kl	length in bp to split the sequence into
chr	chromosome of the sequence to print
position	bp position of the sequence to print

Value

Returns a 6 column data frame with "chr" "pos" "ref.base" "var.base" "ref.seq" "var.seq"

Examples

```
GetPossibleMutations(sequence=c("AGGGATACGTAGACGGTGTAA"),
                     kl=7, chr="chrX", position=1345990)
```

GetSFR

Description

Wrapper function to get the SFR ratio. If indicated and available, use the present vocabulary file to directly grep the SFR. Else get the average profile, estimate the borders and calculate the SFR. Note that for using the vocabulary file only nonambivlent DNA chars are allowed. For the alternative ambivalent chars are decoded.
If preload = TRUE and if vocab.flag = FALSE, preload.profiles has to be submitted. If vocab.flag = TRUE with preload = TRUE then preload.vocab has to be submitted.

Usage

```
GetSFR(kmer,
       tissue,
       data.dir,
       pnorm.tag,
       vocab.flag=FALSE,
       vocab.file=paste0(data.dir,"/",tissue,"/vocabulary_",tissue,".txt"),
       frag.type="",
       preload=FALSE,
       preload.vocab="",
       preload.profiles="")
```

Arguments

kmer	input kmer, FASTA characters string 5 - 7 bp
tissue	name of tissue directory of interest
data.dir	path to directory containing preprocessed data
pnorm.tag	tag indicating which background propensities were used for normalisation
vocab.flag	{FALSE, TRUE} flag if a precalculated vocabulary file is present and should be used [default=FALSE] Speeds up the analysis but precalculation is time consuming, therefore has to be done manually after initial processing of new tissue data. Recommended if tissue is frequently used for analysis.
vocab.file	path to precalculated vocabulary file [default=paste0(data.dir,"",tissue,"vocabulary_",tissue,".txt")]
frag.type	{"DNase", "ATAC"} fragmentation type.
preload	flag [FALSE/TRUE] if to use preloaded average profiles [default=FALSE]
preload.vocab	preloaded vocabulary data frame from PreLoadVocab()
preload.profiles	two dataframe list with the preloaded profiles from PreLoadProfiles()

Value

Returns SFR as sinlge numeric value.

Examples

```
GetSFR(kmer="CACGTG",
       tissue="blood_tissue",
       pnorm.tag="h_ery_1",
       data.dir="./my_sasq_database/human/DNase/",
       vocab.flag=TRUE,
       frag.type="DNase")
```

```
# With preload
data.dir <- "./my_sasq_database/human/DNase/"
tissue <- "blood_tissue"
vocabulary <- PreLoadVocab(data.dir, tissue)

GetSFR(kmer="CACGTG",
       tissue=tissue,
       data.dir=data.dir,
       pnorm.tag="h_ery_1",
       vocab.flag=TRUE,
       frag.type="DNase",
       preload=TRUE,
       preload.vocab=vocabulary)
```

GrepProfile

Description

Grep strand-specific profile of 250 bp surrounding kmer and normalise for total cuts in 250 bp window.

Usage

```
GrepProfile(kmer, infile, preload=FALSE, preload.profiles.strand="")
```

Arguments

kmer	input kmer, FASTA characters string 5 - 7 bp
infile	path to the input file
preload	flag: [FALSE/TRUE] if to use preloaded average profiles [default=FALSE]
preload.profiles.strand	dataframe of preloaded profiles from a single strand [e.g. profiles$plus]

Value

Returns list of **..$profile** and **..$count**.

Examples

```
frag.type <- "DNase"
data.dir <- "./my_sasq_database/human/DNase/"
tissue <- "blood_tissue"
pnorm.tag <- "h_ery_1"
kl <- 7

infile.plus=file.path(data.dir, tissue, "counts",
                      paste0("kmers_", kl, "_count_",
                      tissue, "_pnorm_", pnorm.tag, "_plus.txt"))

profile.plus <- GrepProfile(kmer="CACGTG", infile=infile)
```

```
# With preloading
profiles <- PreLoadKmerProfiles(7, data.dir, tissue, pnorm.tag)

profile.plus <- GrepProfile(kmer="CACGTG", infile=infile, preload=TRUE,
                            preload.profiles.strand=profiles$plus )
```

InSilicoMutation

Description

Wrapper for maximum/absolute damage insilico mutation. Takes an input sequence, splits into data frame of desried kmer length matching window sizes for running comparison and compares reference sequence against all possible mutated sequences (single base pair substitutions). Reports according to report mode ("all", "max", "maxabs").
Note that preloading data is highly recommended for speeding up.
(If preload = TRUE and if vocab.flag = FALSE, preload.profiles has to be submitted. If vocab.flag = TRUE with preload = TRUE then preload.vocab has to be submitted.)

Usage

```
InSilicoMutation(sequence,
      kl=7,
      chr=".",
      position=1,
      report="all",
      damage.mode="exhaustive",
      tissue=tissue,
      data.dir="./my_sasq_database/human/DNase/",
      pnorm.tag,
      vocab.flag=TRUE,
      vocab.file=paste0(data.dir,"/",tissue,"/vocabulary_",tissue,".txt"),
      frag.type=frag.type,
      progress.bar=FALSE,
      preload=FALSE,
      preload.vocab="",
      preload.profiles="")
```

Arguments

sequence1	input sequence, character string
kl	length of the kmer windows, to split up the sequence
chr	chromosome of the sequence to print
position	bp position of the sequence to print, first mutated base is the kl'th base. Therefore input +- 6 bp positions of sequence surounding your sequence of interest and set postion to the kl'th index.
report	{"all", "max", "maxabs"} Select which damage per position to report. [default="all"] **"all"** = report all 3 possible substitutions per position. Reports three rows per bp position. **"max"** = only report substitution with highest positive damage. Report one row per bp positon, easy to convert to wig. **"maxabs"** = only report substitution with highest absolute damage. Reports one row per bp positon as well.
damage.mode	{"exhaustive", "local"} mode to calculate the total damage for the comparison. **"exhaustive"** sum over all pair-wise comparison. **"local"** extract the highest pair-wise damage score. [default="exhaustive"]
tissue	name of tissue directory of interest
data.dir	path to directory containing preprocessed data
pnorm.tag	tag indicating which background propensities were used for normalisation
vocab.flag	{FALSE, TRUE} flag if a precalculated vocabulary file is present and should be used [default=FALSE] Speeds up the analysis but precalculation is time consuming, therefore has to be done manually after initial processing of new tissue data. Recommended if tissue is frequently used for analysis.
vocab.file	path to precalculated vocabulary file [default=paste0(data.dir,"",tissue,"vocabulary_",tissue,".txt")]
frag.type	{"DNase", "ATAC"} fragmentation type.
progress.bar	{FALSE, TRUE} Select if to display a progress.bar when running. (Requires packages pbapply if set to TRUE!) [default=FALSE]
preload	flag [FALSE/TRUE] if to use preloaded average profiles [default=FALSE]
preload.vocab	preloaded vocabulary data frame from PreLoadVocab()
preload.profiles	two dataframe list with the preloaded profiles from PreLoadProfiles()

Value

Seven columns dataframe c(chr, position, ref.base, var.base, ref.sequence, var.sequence, damage).

Examples

```
# use (for example) the BSgenome package to extract reference genome sequence
library(BSgenome)
library(BSgenome.Hsapiens.UCSC.hg18)
genome <- BSgenome.Hsapiens.UCSC.hg18

#set the sequence coordinates of the desired genomic location
chr <- "chr16"
start.pos <- 145852
end.pos <- start.pos + 30

# Get the sequence sequence
seq <- as.character(getSeq(genome, "chr16", start=start.pos-6, end=end.pos+6))
```

```
data.dir <- "./my_sasq_database/human/DNase/"
tissue <- "blood_tissue"
vocabulary <- PreLoadVocab(data.dir, tissue)

#perform the in silcio mutation
df.insilico <- InSilicoMutation(sequence=seq,
                                kl=7,
                                chr="chr16",
                                position=start.pos,
                                report="all",
                                damage.mode="exhaustive",
                                tissue=tissue,
                                pnorm.tag="h_ery_1",
                                data.dir=data.dir,
                                vocab.flag=TRUE,
                                frag.type=frag.type,
                                progress.bar = TRUE,
                                preload=TRUE,
                                preload.vocab=vocabulary
                                )
#display as InSilicoMutationplot
rp <- InSilicoMutationPlot(df.insilico, ylim=c(-4,4))
```

InSilicoMutationPlot

Description

Make a InSilicoMutation plot from data frame as output from InSilicoMutation. Must have been run with report="all"

Usage

```
InSilicoMutationPlot(df, ylim=c(-2,2))
```

Arguments

df	data frame as output from InSilicoMutation (report="all")
ylim	y-limits for plot[default=c(-2,2)]

Value

Returns ggplot2 plot object containing the InSilicoMutationplot.

Examples

```
# use (for example) the BSgenome package to extract reference genome sequence
library(BSgenome)
library(BSgenome.Hsapiens.UCSC.hg18)
genome <- BSgenome.Hsapiens.UCSC.hg18
```

```
#set the sequence coordinates of the desired genomic location
chr <- "chr16"
start.pos <- 145852
end.pos <- start.pos + 30

# Get the sequence sequence
seq <- as.character(getSeq(genome, "chr16", start=start.pos-6, end=end.pos+6))

#perform the i silcio mutation
df.insilico <- InSilicoMutation(sequence=seq,
                                kl=7,
                                chr="chr16",
                                position=start.pos,
                                report="all",
                                damage.mode="exhaustive",
                                tissue="blood_tissue",
                                data.dir="./my_sasq_database/human/DNase/",
                                vocab.flag=TRUE,
                                frag.type=frag.type,
                                progress.bar = TRUE
                                )
#display as InSilicoMutationplot
rp <- InSilicoMutationPlot(df.insilico, ylim=c(-4,4))
```

MakeInSilicoMutationTrackHub

Description

Wrapper to make a UCSC browser track hub from the insilico mutation data frame. Input is a data frame in the same format as the output data frame of the InSilicoMutation function. 7 columns: chr pos ref.base var.base ref.seq var.seq damage

Usage

```
MakeInSilicoMutationTrackHub(input.df,
                     id.tag = "SasQ_RP",
                     store.tracks = paste0("~/", id.tag, "_tracks"),
                     store.hub = paste0("~/", id.tag,"_track_hub"),
                     genome.build = "hg19",
                     path.chr.sizes,
                     short.label = "SasQ In silico mutation InSilicoMutation plot",
                     long.label = "",
                     set.email = "none",
                     bedgraph.to.bigwig.path,
                     make.softlinks = FALSE)
```

Arguments

input.df	input 7 column data frame. chr pos ref.base var.base ref.seq var.seq damage
id.tag	id tag to name the hub directory and bw tracks.
store.tracks	path to directory to store the bigwig tracks.
store.hub	were to store the hub & visualization folder.
genome.build	select genome build ("hg19", "hg18", "mm9", ...) [default="hg19"]
path.chr.sizes	full.path to chrsizes file matching to the selected genome. Not provided in package.
short.label	shortLabel for track hub.
long.label	longLabel for track hub [default = short.label]
set.email	email contact address to appear in trackHub. [default="none"]
bedgraph.to.bigwig.path	full path to UCSC bedGraphtoBigWig convertion tool. Not included.
make.softlinks	{FALSE, TRUE} set flag if to directly make softlinks in hub folder [default=FALSE] (e.g. set TRUE if running directly on a cluster so that the final softlinks paths are already correct [default = FALSE] if data files will be copied to a different direction afterwards (e.g. when mounted and ran locally) if FALSE: After creation copy data hub to desired location and create softlinks in the hub folder to the tracks in the track folder e.g. "ln -s store.tracks/*.bw store.hub"

Value

Writes bigWig tracks into desired dirctory and creates a trackHub structure to migrate to public domain and import to UCSC.

Examples

```
frag.type <- "DNase"
data.dir <- "./my_sasq_database/human/DNase/"
tissue <- "blood_tissue"

# use (for example) the BSgenome package to extract reference genome sequence
library(BSgenome)
library(BSgenome.Hsapiens.UCSC.hg18)
genome <- BSgenome.Hsapiens.UCSC.hg18

#set the sequence coordinates of the desired genomic location
chr <- "chr16"
start.pos <- 145852
end.pos <- start.pos + 30

# Get the sequence sequence
seq <- as.character(getSeq(genome, "chr16", start=start.pos-6, end=end.pos+6))

#perform the i silcio mutation
df.insilico <- InSilicoMutation(sequence=seq,
                                kl=7,
                                chr="chr16",
                                position=start.pos,
```

```
                                    report="all",
                                    damage.mode="exhaustive",
                                    tissue="blood_tissue",
                                    data.dir=data.dir,
                                    vocab.flag=TRUE,
                                    frag.type=frag.type,
                                    progress.bar = TRUE
                                    )

out.dir <- "./myoutputpath/"

# write InSilicoMutationplot ucsc track hub
MakeInSilicoMutationTrackHub(
  input.df = df.insilico,
  id.tag = "SasQ_hub_blood",
  store.tracks = paste0(out.dir, "/", id.tag, "_tracks"),
  store.hub = paste0(out.dir, "/", id.tag,"_track_hub"),
  genome.build = "hg19",
  path.chr.sizes = "~/mydatabase/chrom_sizes/hg19_chrom_sizes.txt",
  short.label = "SasQ In silico mutation InSilicoMutation plot",
  bedgraph.to.bigwig.path = "~/mytools/tools/bedGraphToBigWig",
  make.softlinks = FALSE
)
```

PlotOverlap

Description

Plot two average profiles overlapping or on top of each other given two input profiles. Plot the shoulders if plot.shoulders is set to TRUE, then use shoulders list as provided or determine.

Usage

```
PlotOverlap(profile1,
            profile2,
            kmer1,
            kmer2,
            count1 = "NA",
            count2 = "NA",
            ymode = "separate",
            ylim = c(0,0.01),
            xlim = c(-125,125),
            plot.shoulders=FALSE,
            shoulders1=FALSE,
            shoulders2=FALSE)
```

Arguments

profile1	input profile1, retrieved from the GetFootprint() function as list entry ...$profile
profile2	input profile2, retrieved from the GetFootprint() function as list entry ...$profile
kmer1	input k-mer 1 (reference)
kmer2	input k-mer 2 (variant)
count1	count of k-mer1 occurence
count2	count of k-mer2 occurence
ymode	mode how to plot the overlapping profiles ("merged" or as "separate" profiles above each other) [default=separate]
ylim	ylim to fix for plot [default c(0, 0.01))]
xlim	xlim to fix for plot [default c(-125, 125)]
plot.shoulders	FALSE, TRUE if to plot the estimated shoulders with the profiles [default=FALSE] note that it is only plotted if the separate profile option was selected to keep the plots tidy
shoulders1	list object of estiamted shoulder postion and ranges for profile 1 [default=FALSE]
shoulders2	list object of estiamted shoulder postion and ranges for profile 2 [default=FALSE]

Value

Returns ggplot2 plot object of the overlay plot.

Examples

```
frag.type <- "DNase"
data.dir <- "./my_sasq_database/human/DNase/"
tissue <- "blood_tissue"

kmer1 <- "WGATAA" #note FASTA ambigous code is supported
kmer2 <- "WGATTA"

fp1 <- GetFootprint(kmer=kmer1,
                    tissue=tissue,
                    data.dir=data.dir,
                    frag.type=frag.type,
                    smooth=TRUE)
fp2 <- GetFootprint(kmer=kmer2,
                    tissue=tissue,
                    data.dir=data.dir,
                    frag.type=frag.type,
                    smooth=TRUE)

# make an overlap plot
PlotOverlap(
  fp1$profile,
  fp2$profile,
  kmer1,
  kmer2,
  fp1$count,
  fp2$count,
  ymode="separate"
  )
```

PlotOverlapKmers

Description

Wrapper function to produce an overlay plot from two kmers and a tissue input only.
(For plotting: If preload = TRUE preload.profiles has to be submitted.)

Usage

```
PlotOverlapKmers(kmer1,
                 kmer2,
                 tissue1,
                 tissue2,
                 data.dir,
                 pnorm.tag,
                 frag.type,
                 smooth=TRUE,
                 ylim=c(0,0.01),
                 xlim=c(-125,125),
                 plot.shoulders=FALSE,
                 preload=FALSE,
                 preload.profiles)
```

Arguments

kmer1	input k-mer 1 (reference)
kmer2	input k-mer 2 (variant)
tissue1	name of tissue 1 directory of interest
tissue2	name of tissue 2 directory of interest
data.dir	path to directory containing preprocessed data
pnorm.tag	tag indicating which background propensities were used for normalisation
frag.type	{"DNase", "ATAC"} fragmentation type.
smooth	{FALSE TRUE} if to smooth the profiles. [default=TRUE]
ymode	mode how to plot the overlapping profiles ("merged" or as "separate" profiles above each other) [default=separate]
ylim	ylim to fix for plot [default c(0, 0.01))]
xlim	xlim to fix for plot [default c(-125, 125)]
plot.shoulders	{FALSE, TRUE} if to plot the estimated shoulders with the profiles [default=FALSE] note that it is only plotted if the separate profile option was selected to keep the plots tidy
preload	flag [FALSE/TRUE] if to use preloaded average profiles [default=FALSE]
preload.profiles	two dataframe list with the preloaded profiles from PreLoadProfiles()

Value

Returns ggplot2 plot object of the overlay plot.

Examples

```
frag.type <- "DNase"
data.dir <- "./my_sasq_database/human/DNase/"
tissue <- "blood_tissue"

PlotOverlapKmers(
  kmer1="CACGTG",
  kmer2="CACGTT",
  tissue1=tissue,
  issue2=tissue,
  data.dir=data.dir,
  pnorm.tag="h_ery_1",
  frag.type="DNase",
  smooth=TRUE,
  plot.shoulders=FALSE
  )
```

PlotSingle

Description

Plot the average profile plot given an input profile. Plot the shoulders if plot.shoulders is set to TRUE and use shoulders list provided or determine.

Usage

```
PlotSingle(profile,
           kl=7,
           plot.shoulders=FALSE,
           shoulders=FALSE,
           ylim=c(0,0.01),
           xlim=c(-125,125),
           color="black")
```

Arguments

profile	input profile
kl	length of the kmer windows, to split up the sequences
ylim	ylim to fix for plot [default=c(0, 0.01))]
xlim	xlim to fix for plot [default=c(-125, 125)]
plot.shoulders	{FALSE, TRUE} if to plot the estimated shoulders with the profiles [default=FALSE]
shoulders	list object of estiamted shoulder postion and ranges for profile 1 [default=FALSE]
color	Select a color for the profile [default="black"]

Value

Returns single ggplot2 plot object of the profile.

Examples

```
frag.type <- "DNase"
data.dir <- "./my_sasq_database/human/DNase/"
tissue <- "blood_tissue"
kmer <- "CACGTG"

# get the footprint
fp <- GetFootprint(kmer=kmer, tissue=tissue, data.dir=data.dir,
                   frag.type=frag.type, smooth=TRUE)

# estimate the shoulders from the profile
# (use smoothed profile or smooth within call!)
sh <- SobelBorders(fp$profile, kl=nchar(kmer))

# make single, merged profile plot
p <- PlotSingle(profile=fp$profile,
                kl=nchar(kmer),
                plot.shoulders=TRUE,
                shoulders=sh
                )
```

PlotSingleKmer

Description

Wrapper function to produce a plot from kmer and tissue input only.
(For plotting: If preload = TRUE preload.profiles has to be submitted.)

Usage

```
PlotSingleKmer(kmer,
               tissue,
               data.dir,
               frag.type,
               pnorm.tag,
               smooth=TRUE,
               smooth.bandwidth=5,
               plot.shoulders=FALSE,
               ylim=c(0,0.01),
               xlim=c(-125,125),
               color="black",
               preload=FALSE,
               preload.profiles)
```

Arguments

kmer	input k-mer
tissue	name of tissue 1 directory of interest
data.dir	path to directory containing preprocessed data
pnorm.tag	tag indicating which background propensities were used for normalisation
frag.type	{"DNase", "ATAC"} fragmentation type.
smooth	{FALSE TRUE} if to smooth the profiles. [default=TRUE]
smooth.bandwidth	bandwidth to smooth in bp [default=5]
plot.shoulders	{FALSE, TRUE} if to plot the estimated shoulders with the profiles [default=FALSE]
ylim	ylim to fix for plot [default c(0, 0.01))]
xlim	xlim to fix for plot [default c(-125, 125)]
color	Select a color for the profile [default="black"]
preload	flag [FALSE/TRUE] if to use preloaded average profiles [default=FALSE]
preload.profiles	two dataframe list with the preloaded profiles from PreLoadProfiles()

Value

Returns single ggplot2 plot object of the profile.

Examples

```
PlotSingleKmer(kmer="CACGTG",
               tissue="blood_tissue",
               data.dir="./my_sasq_database/human/DNase/",
               frag.type="DNase")
```

PlotSingleStrands

Description

Wrapper function to produce strand specific plots from kmer and tissue input only.

Usage

```
PlotSingleStrands(kmer,
                  tissue,
                  data.dir,
                  pnorm.tag,
                  frag.type,
                  smooth=TRUE,
                  smooth.bandwidth=5,
                  background.flag=FALSE,
                  ylim=c(0,0.01),
                  xlim=c(-125,125))
```

Arguments

kmer	input k-mer
tissue	name of tissue 1 directory of interest
data.dir	path to directory containing preprocessed data
pnorm.tag	tag indicating which background propensities were used for normalisation
frag.type	{"DNase", "ATAC"} fragmentation type.
smooth	{FALSE TRUE} if to smooth the profiles. [default=TRUE]
smooth.bandwidth	bandwidth to smooth in bp [default=5]
background.flag	FALSE TRUE Select if to visualize the genome-wide background cut profiles or from tissue data. Will adjust the required and queried data structure accordingly. Note that if set to TRUE, a different data directory, that pointing to the storange of the deproteinized, genome-wide background has to be provied. [default=FALSE]
ylim	ylim to fix for plot [default c(0, 0.01))]
xlim	xlim to fix for plot [default c(-125, 125)]

Value

Returns list of strand-specific profile plots: for plus **...$plot.plus** and minus strand **...$plot.minus**.

Examples

```
PlotSingleStrands(kmer="WGATAA",
                  tissue = "blood_tissue",
                  data.dir = "./my_sasq_database/human/DNase/",
                  pnorm.tag = "h_ery_1",
                  frag.type = "DNase")
```

PreLoadProfiles

Description

Preload the entire kmer based average profiles for a given tissue and kmer size.

Usage

```
PreLoadKmerProfiles(kl, data.dir, tissue, pnorm.tag)
```

Arguments

kl	length of the kmer windows, to split up the sequences
tissue	name of tissue 1 directory of interest
data.dir	path to directory containing preprocessed data
pnorm.tag	tag indicating which background propensities were used for normalisation

Value

List of legnth two ...$plus and ...$minus storing the occurrences and cut profiles per k-mer. Each list entry is a data frame with the following dimension:
Column[1] = kmer, Column[2] = kmer occurrence, Column[3 - (300+kl+2)] = 300 bp + kl kmer surrounding profile. Rows = number of possible kmers of length kl
Note that we store the 300 kmer surrounding bases but only query and process the surrounding 250 bp window. We kept the stroing at 300 bp to be quickly able to expand our scripts. All functions process the described 250 surrounding window from this start files.

Examples

```
kl <- 7

data.dir <- "./my_sasq_database/human/DNase/"

tissue <- "blood_tissue"

pnorm.tag <- "h_ery_1"

profiles <- PreLoadProfiles(data.dir, tissue)
```

PreLoadVocab

Description

Preload the entire tissue specific vocabulary file with preprocessed SFRs for every possible k-mer.

Usage

```
PreLoadVocab(data.dir, tissue)
```

Arguments

tissue	name of tissue 1 directory of interest
data.dir	path to directory containing preprocessed data

Value

Vocabulary file loaded as two column data frame.

Examples

```
data.dir <- "./my_sasq_database/human/DNase/"

tissue <- "blood_tissue"

vocabulary <- PreLoadVocab(data.dir, tissue)
```

PruneProfile

Description

Prune an retrieved (250 bp) average profile equally from both directions given the profile and the desired window size around the kmer which to retrieve. Requires an even number as length to prune. Length of the output profile will always be (desired.length + kmer.length).

Usage

```
PruneProfile(profile, desired.length)
```

Arguments

profile	input profile
desired.length	length of to prune to (will be desried.length + kmer.length)

Value

Returns pruned profile as numeric vector.

Examples

```
frag.type <- "DNase"
data.dir <- "./my_sasq_database/human/DNase/"
tissue <- "blood_tissue"
kmer <- "CACGTG"

# get the footprint
fp <- GetFootprint(kmer=kmer, tissue=tissue, data.dir=data.dir,
                   frag.type=frag.type, smooth=TRUE)

#prune
pruned.profile <- PruneProfile(fp$profile, 100)
```

QueryJaspar

Description

Take a sequence input and query it against a set of Jaspar2014 PWMs. (as provided or saved from the JASPAR2014 R package). Requires "Biostrings" and "TFBSTools" packages.

Usage

```
QueryJaspar(sequence, threshold=0.8, pwm.data)
```

Arguments

sequence	input sequence
threshold	relative percentage score threshold above which to report matches [default=0.8]
pwm.data	a stored pwm.RData object as provided with the Sasquatch distriution or as retrieved and saved from JASPAR2014 R package

Value

Returns character string listing the PWM matches above the relative percentage score threshold.

Examples

```
#requries Biostrings and TFBSTools R packages
library(Biostrings)
library(TFBSTools)

#load human.pwm object
load("./my_sasq_database/jaspar/jaspar2014.human.9606.all.versions")

# Single JASPAR query
QueryJaspar(sequence="AGATAATAG", threshold=0.8, pwm.data=human.pwm)
```

QueryJasparBatch

Description

Take a data frame from the RefVarBatch query as input and query it against the set of selected Jaspar2014 PWMs using a selected match.threshold. Select an absolute footprinting damage above which to query jaspar.

Usage

```
QueryJasparBatch(df, damage.threshold=0, match.threshold=0.8, pwm.data)
```

Arguments

df	9 column data frame as output from RefVarBatch()
damage.threshold	absolute predicted footprinting damage above which a sequence should be selected for the query. [default=0, query all]
match.threshold	relative percentage score threshold above which to report matches [default=0.8]
pwm.data	a stored pwm.RData object as provided with the Sasquatch distriution or as retrieved and saved from JASPAR2014 R package

Value

Returns data frame with additional column for jaspar query results.

Examples

```
#ty example data frame
tdf <- data.frame(
        id=c("1", "2", "3"),
        ref=c("ATAGATAATCGCT", "ATAGATAATCGCT", "ATATATTCTCGCT"),
        var=c("ATAGATCATCGCT", "ATAGATTATCGCT", "ATAGATGATCGCT")
        )

#make a reference vs. variant batch query
comp.df <- RefVarBatch(ref.var.df=tdf,
                       kl=7,
                       damage.mode="exhaustive",
                       tissue="blood_tissue",
                       data.dir="./my_sasq_database/human/DNase/",
                       vocab.flag=TRUE,
                       frag.type="DNase")

#requries Biostrings and TFBSTools R packages
library(Biostrings)
library(TFBSTools)

#load human.pwm object
load("./my_sasq_database/jaspar/jaspar2014.human.9606.all.versions")

#query refvar data frame against jaspar pwms
comp.df.jaspar <- QueryJasparBatch(df=comp.df,
                                   damage.threshold=0.3,
                                   match.threshold=0.8,
                                   pwm.data=human.pwm)
```

QueryLongSequence

Description

Wrapper function to split a longer sequence into kmers of length kl and return kmer, SFR and plots if specified. (If preload = TRUE and if vocab.flag = FALSE, preload.profiles has to be submitted. If vocab.flag = TRUE with preload = TRUE then preload.vocab has to be submitted.)

Usage

```
QueryLongSequence(sequence,
                  kl,
                  tissue,
                  data.dir,
                  pnorm.tag,
                  vocab.flag=FALSE,
                  vocab.file=paste0(data.dir,"/",tissue,
                                    "/vocabulary_",tissue,".txt"),
                  frag.type,
                  plots=FALSE,
                  smooth=TRUE,
```

```
                  plot.shoulders=TRUE,
                  ylim=c(0,0.01),
                  xlim=c(-125,125),
                  preload=FALSE,
                  preload.vocab="",
                  preload.profiles="")
```

Arguments

sequence	input sequence, character string
kl	length of the kmer windows, to split up the sequence
tissue	name of tissue directory of interest
data.dir	path to directory containing preprocessed data
pnorm.tag	tag indicating which background propensities were used for normalisation
vocab.flag	{FALSE, TRUE} flag if a precalculated vocabulary file is present and should be used [default=FALSE] Speeds up the analysis but precalculation is time consuming, therefore has to be done manually after initial processing of new tissue data. Recommended if tissue is frequently used for analysis.
vocab.file	path to precalculated vocabulary file [default=paste0(data.dir,"",tissue,"vocabulary_",tissue,".txt")]
frag.type	{"DNase", "ATAC"} fragmentation type.
plots	{FALSE, TRUE} indicate if to plot a profile per kmer [default=FALSE]
smooth	{FALSE, TRUE} smooth the profile [default=TRUE]
plot.shoulders	{FALSE, TRUE} print the estimated shoulders with the plots [default=TRUE]
ylim	y-limit for plots [default c(0,0.01)]
xlim	x-limit for plots [default c(-125,125)]
preload	flag [FALSE/TRUE] if to use preloaded average profiles [default=FALSE]
preload.vocab	preloaded vocabulary data frame from PreLoadVocab()
preload.profiles	two dataframe list with the preloaded profiles from PreLoadProfiles()

Value

Returns **...$df** a data frame listing the splitted kmers with the respective SFR.
...$plots if specified list of profile plots with one plot per splitted k-mer.

Examples

```
QueryLongSequence(sequence="ATAGATAATCGCT",
                  kl,
                  tissue="blood_tissue",
                  pnorm.tag="h_ery_1",
                  data.dir="./my_sasq_database/human/DNase/",
                  vocab.flag=FALSE,
                  vocab.file=paste0(data.dir,"/",tissue,
                                    "/vocabulary_",tissue,".txt"),
                  frag.type="DNase",
                  plots=FALSE)

# With preload
data.dir <- "./my_sasq_database/human/DNase/"
tissue <- "blood_tissue"
vocabulary <- PreLoadVocab(data.dir, tissue)
```

```
QueryLongSequence(sequence="ATAGATAATCGCT",
                  kl,
                  tissue=tissue,
                  pnorm.tag="h_ery_1",
                  data.dir=data.dir,
                  vocab.flag=TRUE,
                  vocab.file=paste0(data.dir,"/",tissue,
                                    "/vocabulary_",tissue,".txt"),
                  frag.type="DNase",
                  plots=FALSE,
                  preload=TRUE,
                  preload.vocab=vocabulary)
```

RefVarBatch

Description

Wrapper function to analyse multiple Ref-Var-Sequence pairs. Split each sequence into kmers of length kl, get their SFRs (ideally for speed from vocab.file or calculate per instance. Calculate the damage associated with each kmer pair and from that the local or the exhaustive summed up damage of entire sequence pair.
(If preload = TRUE and if vocab.flag = FALSE, preload.profiles has to be submitted. If vocab.flag = TRUE with preload = TRUE then preload.vocab has to be submitted.)

Usage

```
RefVarBatch(ref.var.df,
            kl,
            damage.mode="exhaustive",
            tissue,
            data.dir,
            pnorm.tag,
            vocab.flag=FALSE,
            vocab.file=paste0(data.dir,"/",tissue,"/vocabulary_",tissue,".txt"),
            frag.type,
            preload=FALSE,
            preload.profiles="",
            preload.vocab="")
```

Arguments

ref.var.df	three column data frame listing id reference and variance sequence (id reference variant)
kl	length of the kmer windows, to split up the sequences
damage.mode	{"exhaustive", "local"} mode to calculate the total damage for the comparison. **"exhaustive"** sum over all pair-wise comparison. **"local"** extract the highest pair-wise damage score. [default="exhaustive"]
tissue	name of tissue directory of interest
data.dir	path to directory containing preprocessed data
pnorm.tag	tag indicating which background propensities were used for normalisation
vocab.flag	{FALSE, TRUE} flag if a precalculated vocabulary file is present and should be used [default=FALSE] Speeds up the analysis but precalculation is time consuming, therefore has to be done manually after initial processing of new tissue data. Recommended if tissue is frequently used for analysis.
vocab.file	path to precalculated vocabulary file [default=paste0(data.dir,"",tissue,"vocabulary_",tissue,".txt")]
frag.type	{"DNase", "ATAC"} fragmentation type.
preload	flag [FALSE/TRUE] if to use preloaded average profiles [default=FALSE]
preload.vocab	preloaded vocabulary data frame from PreLoadVocab()
preload.profiles	two dataframe list with the preloaded profiles from PreLoadProfiles()

Value

Returns data frame listing Ref and Var sequence with highest scoring kmer pair and according SFRs and calculated (exhaustive or local) total damage.

Examples

```
#ty example data frame
tdf <- data.frame(
        id=c("1", "2", "3"),
        ref=c("ATAGATAATCGCT", "ATAGATAATCGCT", "ATATATTCTCGCT"),
        var=c("ATAGATCATCGCT", "ATAGATTATCGCT", "ATAGATGATCGCT")
        )

# Make a reference vs. variant batch query
comp.df <- RefVarBatch(ref.var.df=tdf,
                       kl=7,
                       damage.mode="exhaustive",
                       tissue="blood_tissue",
                       data.dir="./my_sasq_database/human/DNase/",
                       pnorm.tag="h_ery_1",
                       vocab.flag=TRUE,
                       frag.type="DNase")

# With Preload
data.dir <- "./my_sasq_database/human/DNase/"
tissue <- "blood_tissue"
vocabulary <- PreLoadVocab(data.dir, tissue)
comp.df <- RefVarBatch(ref.var.df=tdf,
                       kl=7,
                       damage.mode="exhaustive",
                       tissue="blood_tissue",
```

```
                    data.dir="./my_sasq_database/human/DNase/",
                    pnorm.tag="h_ery_1",
                    vocab.flag=TRUE,
                    frag.type="DNase",
                    preload=TRUE,
                    preload.vocab=vocabulary)
```

SmoothProfile

Description

Helper function to smooth a profile given the specified badnwidth and a gaussian, normal kernel.

Usage

```
SmoothProfile(profile, bandwidth=5)
```

Arguments

profile	input profile (numeric vector)
smooth.bandwidth	bandwidth to smooth in bp [default=5]

Value

Returns smoothed profile as numeric vector.

Examples

```
fp <- GetFootprint(kmer="CACGTG",
                   tissue="blood_tissue",
                   data.dir="./my_sasq_database/human/DNase/",
                   frag.type="DNase",
                   smooth=FALSE)

smoothed.profile <- SmoothProfile(fp$profile, 5)
```

SobelBorders

Description

Estimate the footprint shoulders by based on zero crossings of the 1D 1st derivative approximation of a smoothed profile and get the optimal shoulder range that maximizies the SFRatio of the footprint. (Approximations for speeding up the process are: Estimate the optimal positions based on 6 bp wide shoulders. Then opimize the shoulder width/range with allowed ranges in {4,6,8,10})

Usage

```
SobelBorders(profile, kl)
```

Arguments

profile	input profile (numeric vector) has to be smoothed for proper function!
kl	{5,6,7} kmer length input

Value

Returns list object containing the shoulder centric positions and their ranges.
..$us upstream shoulder center
..$ds downstream shoulder center
..$us.range width/range of upstream shoulder
..$ds.range width/range of downstream shoulder
..$flag flag {TRUE FALSE} if shoulders could be estimated

Examples

```
# get the footprint
fp <- GetFootprint(kmer="CACGTG",
                   tissue="blood_tissue",
                   data.dir="./my_sasq_database/human/DNase/",
                   frag.type="DNase",
                   smooth=TRUE)

# estimate the shoulders from the profile
# (use smoothed profile or smooth within call!)
sh <- SobelBorders(fp$profile, kl=nchar(kmer))
```

Sobeln

Description

Helper function to calculate 1D 1st derivative approximation of profile by 1D sobel filtering.

Usage

```
Sobeln(profile)
```

Arguments

profile	input profile (numeric vector) should be smoothed!

Value

Returns 1D 1st derivative approimation of the profile as numeric vector.

Examples

```
fp <- GetFootprint(kmer="CACGTG",
                   tissue="blood_tissue",
                   data.dir="./my_sasq_database/human/DNase/",
                   frag.type="DNase",
                   smooth=TRUE)

# estimate the shoulders from the profile
# (use smoothed profile or smooth within call!)
approx.1st.deriv.prof<- Sobeln(fp$profile)
```