

GeneR Package

L. Cottret, A. Lucas, O. Rogier, E. Marrakchi, V. Lefort,
P. Durosay, A. Viari, C. Thermes & Y. d'Aubenton-Carafa

May 12, 2008

Contents

1	Overview	1
2	Installation	2
3	Quick Start	2
4	Usage	2
4.1	Basique manipulation with sequences as character string	2
4.2	Manipulation of sequences with buffers	3
4.2.1	Introduction	3
4.2.2	Basic manipulation	4
4.2.3	Look for motifs, composition, masked position	5
4.2.4	Adresses manipulations	6
4.3	Bank files	8
4.3.1	Retrieve sequences	8
4.3.2	Fasta files	8
4.3.3	Embl files	9
4.3.4	GenBank files	10
4.4	Segments manipulations	10
4.4.1	Overview	10
4.4.2	Examples	11
4.5	Random sequences	11
4.5.1	Presentation	11
4.5.2	Example	11
4.6	Profiles	14
4.6.1	Overview	14
4.6.2	Examples	14
5	A complete Example	15

1 Overview

To summarize, we can split major functions into 5 categories.

Reading and writing sequences

GeneR has been designed for fast sequence retrieving even from very large sequence databanks, in Fasta, Embl or Genbank formats. It is also possible to enter sequences directly from a R command line.

Handling sequences

The usual copy-paste of parts of sequences or other manipulations can be performed by functions using vectors of strands and positions. Annotations from the features field within formatted sequence entries can be extracted and used directly in vectors. By this way, it is easy to extract sequence fragments of interest.

Analyzing sequences

Some tools are designed to count oligo-nucleotides, to look for exact word positions or to shuffle sequences (useful for statistical validations).

Manipulation of regions on a chromosome

In all the amount of annotations, we often have mass informations on genes (introns, CDS, isoforms) but we have to deduce intergenic regions, exons or more sophisticated regions. We provide a complete set of tools to easily compute any subregions, without an exhaustive texture on a whole chromosome.

Genetic tools

Finally, the package also contains functions related to genetic and structural aspects of the sequences : ORF localization, translation, or RNA secondary structure determination (with extension of GeneR: GeneRfold package).

2 Installation

With package sources, you can try also something like:

```
R CMD INSTALL GeneR_x.x-x.tar.gz
```

or try in the menu (for Windows and mac users).

3 Quick Start

First steps on GeneR could be to check man page of GeneR:

```
library(GeneR)
help(GeneR)
```

And for impatient:

```
demo(GeneR)
```

4 Usage

4.1 Basique manipulation with sequences as character string

Standard R commands allow to extract subpart of a character string, or append two character string, put in upper case (functions *substr*, *paste*, *toupper*). We add tools for the specific use of genetic sequences: insert a sequence into a first one, compute the reverse complementary, count mono, di or tri-nucleotides of sequence and the possibility to import sequence from a Embl, Fasta or GeneBank file.

Example

Insert a poly A into sequence

```
> library(GeneR)
> s <- "gtcatgcatgctaggtgacagttaaaatgcgtctaggtgacagtctaaca"
> s2 <- insertSeq(s, "aaaaaaaaaaaaa", 20)
> s2

[1] "gtcatgcatgctaggtgacaaaaaaaaaaaaaagttaaaatgcgtctaggtgacagtctaaca"
```

Compute the reverse complementary:

```
> library(GeneR)
> strComp(s)

[1] "ttgttagactgtcacctagacgcattttaactgtcacctagcatgcatgac"
```

Count mono nucleotides, di-nucleotides (wsize can be from 1 to 15 if computer has enough memory).

```
> strCompoSeq(s2, wsize = 1)

      T      C      A      G      X
[1,] 0.2 0.1384615 0.4615385 0.2 0

> strCompoSeq(s2, wsize = 2)

      TT TC      TA      TG TX      CT CC      CA CG CX      AT AC      AA      AG
[1,]  0  0 0.03125 0.0625  0 0.09375  0 0.15625  0  0 0.03125  0 0.28125 0.0625
      AX      GT      GC      GA GG GX XT XC XA XG XX
[1,]  0 0.1875 0.03125 0.0625  0  0  0  0  0  0  0  0
```

Get some data from the web

```
> seqNcbi("BY608190", file = "toto.fa")

[1] 1

> strReadFasta("toto.fa", from = 10, to = 35)

[1] "TGCGTTTGT TTTT TAGTGACTTCTAC"
```

4.2 Manipulation of sequences with buffers

4.2.1 Introduction

To work on large sequences (i.e. a whole chromosome), we use a C++ adapted class that requires a minimum memory allocation.

Standard use: a chromosome load in the buffer, all computation done with GeneR functions, and only usefull results returns to standard R objects.

4.2.2 Basic manipulation

Sequence read from a fasta file:

```
> readFasta("toto.fa")
```

```
[1] 0
```

```
> readFasta("toto.fa", seqno = 1)
```

```
[1] 0
```

Show current buffer

```
> getSeq(from = 10, to = 35)
```

```
[1] "TGC GTTTG TTTT TAGTGACTTCTAC"
```

Same previous functions with buffers: Extract from multiple positions

```
> getSeq(seqno = 0, from = c(1, 10, 360), to = c(10, 20, 0))
```

```
[1] "TGGGCTTATT" "TGC GTTTG TTT"
```

```
[3] "AATAAAGCCAACCTGCAGCTGCTGTT"
```

```
> assemble(seqno = 0, destSeqno = 1, from = c(1, 10, 360), to = c(10,
+ 20, 0))
```

```
[1] 1
```

```
> getSeq(seqno = 1)
```

```
[1] "TGGGCTTATTTCGTTTGTTTAATAAAGCCAACCTGCAGCTGCTGTT"
```

See the end of sequence

```
> sizeSeq(seqno = 0)
```

```
[1] 385
```

```
> size0 <- sizeSeq(seqno = 0)
```

```
> getSeq(seqno = 0, from = size0 - 20, to = 0)
```

```
[1] "AGCCAACCTGCAGCTGCTGTT"
```

Append and concat:

```
> getSeq(0, from = 1, to = 35)
```

```
[1] "TGGGCTTATTTCGTTTGTTT TAGTGACTTCTAC"
```

```
> getSeq(1)
```

```
[1] "TGGGCTTATTTCGTTTGTTTAATAAAGCCAACCTGCAGCTGCTGTT"
```

```
> appendSeq(0, 1)
```

```
[1] 0
```

```
> getSeq(seqno = 0, from = size0 - 20, to = size0 + 10)
```

```
[1] "AGCCAACCTGCAGCTGCTGTTTGGGCTTATT"
```

```
> concat(seqno1 = 0, seqno2 = 1, destSeqno = 3, from1 = 2, to1 = 10,
+ from2 = 8, to2 = 0, strand1 = 1)
```

```
[1] 3
```

```
> getSeq(3)
```

```
[1] "AATAAGCCCAACAGCAGCTGCAAGTTGGCTTTATTAACAAACGCAAAT"
```

4.2.3 Look for motifs, composition, masked position

Several tools are designed to

- gets match positions of an oligomer in fragments of a sequence
- gets ORFs (Open Reading Frames) from a sequence.
- Get composition in mono, di or trinucleotides of fragments of a sequence
- mask sequence in lower case or with letter 'N'
- get masked position from a sequence.
- change sequence to Dna or Rna.

Look for motifs

```
> exactWord("AAAG", seqno = 0)
```

```
[[1]]  
[1] -1
```

Find Orfs

```
> getOrfs(seqno = 0)
```

```
NULL
```

```
> maxOrf(seqno = 0)
```

```
[1] -1
```

Mask sequences while lowering case of parts of sequence

```
> x = c(5, 15, 30)  
> y = c(10, 20, 35)  
> lowerSeq(from = x, to = y)
```

```
[1] 1
```

```
> getSeq(from = 1, to = 35)
```

```
[1] "TGGGccttattGCGTttgtttTTTAGTGACttctac"
```

And the reverse: get masked positions from a sequence:

```
> posMaskedSeq(seqno = 0, type = "lower")
```

```
      from to  
[1,]    5 10  
[2,]   15 20  
[3,]   30 35
```

```
> posMaskedSeq(seqno = 0, type = "upper")
```

```
      [,1] [,2]  
[1,]    1   4  
[2,]   11  14  
[3,]   21  29  
[4,]   36 432
```

Or masked with N:

```
> s <- "ATGCtgTGTtagtacATNNNNNNNNNNNNNTGGGTTaAAAattt"
> placeString(s, upper = FALSE, seqno = 0)

[1] 0

> posMaskedSeq(seqno = 0, type = "N")

      from to
[1,]    18 32

Change from Dna to Rna alphabet

> dnaToRna()

[1] 0

> getSeq()

[1] "AUGCugUGUUaguacAUNNNNNNNNNNNNNNUGGGUUUaAAAauuu"
```

4.2.4 Adresses manipulations

Presentation When GeneR load a subset of a larger sequence stored in a bank file, it will store the following informations in the C adapted class (buffers, by default 100 buffers than can be extended if necessary):

- subsequence (i.e. the succession of A,T,G,C).
- postions of the extremities of the subsequence in the master sequence
- size of the whole sequence in the bank file
- name of the sequence

For specific purposes as renaming a sequence, all these variables can be viewed and carefully changed at any time (here functions *getAccn* and *setAccn*).

Several sequences can be stored simultaneously and called by their buffer number.

Strand is another global variable which can be set and viewed (functions *getStrand* and *setStrand*). It is used as input parameter in many functions to analyze complementary strand. It was designed to avoid doing explicitly the complement of the loaded strand then to store it in a buffer with, as consequence, loss of the informations linked to the master sequence.

We have defined 3 types of addresses on a subsequence extracted from a master sequence:

- Absolute addresses i.e. addresses on the master sequence, from the 5' end of the input strand refered as forward (noted A)
- Real addresses, i.e. addresses on the master sequence, from the 5' end of one of strands (noted R)
- Relative addresses, i.e. addresses on working subsequence, from the 5' end of one of strands (noted T).

Let's show an example, if we read sequence from 11 to 25 from a gene of size 40 (see Figure1). Obviously, when an entire sequence is stored, real and relative addresses will be the same.

Although all functions using positions need and return absolute addresses, 6 functions allow to convert R, A, T into any other type (functions *RtoA*, *RtoT*, *AtoR*, *AtoT*, *TtoR*, *TtoA*).

A global variable *strand* is used to convert positions (see *setStrand* *getStrand*).

We keep globals variables of sequences... We implement tools to ...

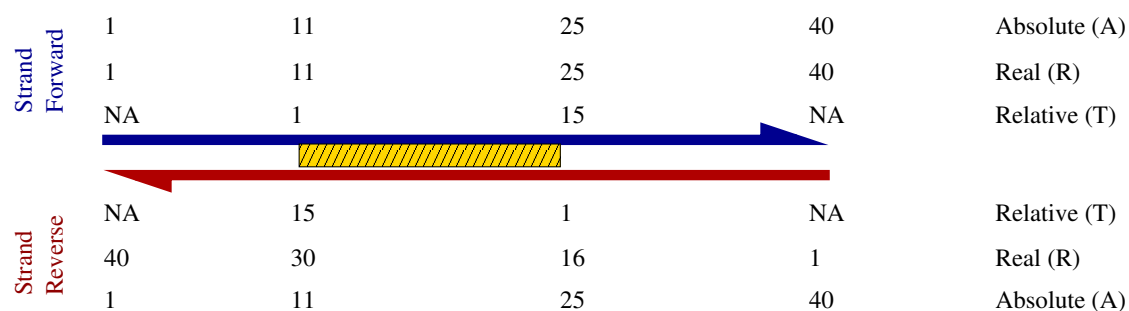


Figure 1: Addresses of a sequence

Example Imagine our chromosome is as follow, and we study only part of chromosome between position 11 to 25:

```
> s <- "xxxxxxxxxxATGTGTCGTTAATTGxxxxxxxxxxxxxxxxx"
> placeString(s)
```

```
[1] 0
```

```
> setStrand(0)
```

```
[1] 0
```

```
> writeFasta(file = "toto.fa")
```

```
[1] 1
```

```
> readFasta(file = "toto.fa", from = 11, to = 25)
```

```
[1] 0
```

```
> getSeq()
```

```
[1] "ATGTGTCGTTAATTG"
```

Orfs on 'Absolute' and 'relative' Adresses

```
> getOrfs()
```

```
NULL
```

```
> AtoT(getOrfs())
```

```
numeric(0)
```

More Fun:

```
> getSeq()
```

```
[1] "ATGTGTCGTTAATTG"
```

```
> exactWord(word = "AA")
```

```
[[1]]
```

```
[1] -1
```

```
> AtoT(exactWord(word = "AA")[[1]])
```

```

[1] NA
> setStrand(1)
[1] 1
> getSeq()
[1] "CAATTAACGACACAT"
> exactWord(word = "AA")
[[1]]
[1] -1
> AtoT(exactWord(word = "AA")[[1]])
[1] NA
> exactWord(word = "TT")
[[1]]
[1] -1
> AtoT(exactWord(word = "TT")[[1]])
[1] NA

```

4.3 Bank files

4.3.1 Retrieve sequences

We develop 2 functions to get sequence files from internet: seqUrl (from a srs server) and seqNcbi (from Ncbi).

Sequence are retrieve in Fasta or Embl format for seqUrl; Fasta or GenBank format for seqNcbi.

```

> seqNcbi("BY608190", file = "bank.fa")
> seqNcbi("BY608190", file = "BY608190.gbk", type = "genbank")

```

4.3.2 Fasta files

Basic tools to get or write informations from bank file

```

> readFasta(file = "bank.fa", name = "gi|26943372|gb|BY608190.1|BY608190",
+   from = 1, to = 30)
[1] 0
> getSeq()
[1] "TGGGCTTATTGCGTTTGTTCCTTTAGTGACT"
> fastaDescription(file = "bank.fa", name = "gi|26943372|gb|BY608190.1|BY608190")
[1] "BY608190 RIKEN full-length enriched, visual cortex Mus musculus cDNA clone K230301C17 3', mR

```

And to write a to the fasta bank file:

```

> s = "cgtagctagctagctagctagctagctagctagcta"
> placeString(s, seqno = 0)

```



```

CC   This is a comment for      this dummy sequence... I try to be long enough
CC   to show that this comment   will be written on several lines
XX
FT   CDS                        <1..12
FT                               /codon_start=2
FT                               /gene="toto"
FT                               /note="Here is      what I think about this gene"
XX
FT                               /translation=""
SQ   Sequence 51 BP; 15 A; 11 C; 13 G; 12 T; 0 other;
      gtcatgcatc ctaggtgtca gggaaaatgc gtctacgtga cagtctaaca a           51
//

```

4.3.4 GenBank files

We have a function to open any sequence from a GeneBank files

```

> readGbk(file = "BY608190.gbk", name = "BY608190", from = 10,
+         to = 30)

[1] 0

> getSeq()

[1] "TGCGTTTGT TTTT TAGTGACT"

```

4.4 Segments manipulations

4.4.1 Overview

We made a set of tools manipulation segments. The aim of these classes is to describe regions on chromosomes that are discontinuous segments on a line like:

	1	10	25	40
Region A:	#####		#####	
Region B:	####	####	#####	
Region C:				#####

We made two kind of class

- **segSet**: segments set, is a matrix nx2 composed of a column of "from", and a column of "to". Used to describe a region like 'A' or 'B' in our example. (A matrix 3x2 to describe region 'B').
- **globalSeg**: a list of **segSet**. It allows the notion of list of discontinuous segments (our use: a list of gene's models as a gene's model is stored as a list of its exons). In our sample, globalSeg will be the list of the 3 regions A,B and C. Note that it store more information than just a matrix with 2 columns containing all **segments** of theses 3 regions.

For a better comprehension of other man pages, we introduce this notation:

- a segment is just a part of a line determined by two values (from and to)
- an object of class **segSet** is a set of **n** segments, determined by a matrix **nx2**
- an object of class **globalSeg** is a set of **segSets**, determined by a list of matrix.

4.4.2 Examples

We show segments usage with figure:

- Figure 2 presents our objects
- Figure 3 show union, intersection, and basic manipulation on object of class *segSet*.
- Figure 4 show union, intersection, and basic manipulation on object of class *globalSeg*.
- Figure 4.4.2 show unary operators on our objects.

4.5 Random sequences

4.5.1 Presentation

We made two tools to generate a random sequence, or to randomize an existing sequence.

Function *randomSeq* creates a random sequence from a distribution of nucleotides, of ploy-nucleotides. A real composition of nucleotides can be use from function *compoSeq*, with param *p=TRUE*.

Function *shuffleSeq* generate a sequence while assembling at random specific number of each nucleotides (or ploy-nucleotides). These number of nucleotide can be provided by function *compoSeq*, with param *p=FALSE*: it is then a re-assemblage of all nucleotides (or tri-nucleotides, or ploy-nucleotides) of a real sequence.

4.5.2 Example

```
> set.seed(3)
> randomSeq(prob = c(0.2, 0.3, 0.2, 0.3), letters = c("T", "C",
+ "A", "G"), n = 30)

[1] "CTGGAACCGAGGGTTCATCCCCCAGTGA"

> randomSeq(prob = rep(0.0625, 16), letters = c("TT", "TC", "TA",
+ "TG", "CT", "CC", "CA", "CG", "AT", "AC", "AA", "AG", "GT",
+ "GC", "GA", "GG"), n = 10)

[1] "CGCATGATCCCAGGCTAACT"

> shuffleSeq(count = c(7, 3, 3, 4, 0), letters = c("T", "C", "A",
+ "G", "N"))

[1] "TATCTTTTGTCTGGACGA"

> shuffleSeq(count = c(rep(4, 4), rep(2, 4), rep(1, 4), rep(0,
+ 4)), letters = c("TT", "TC", "TA", "TG", "CT", "CC", "CA",
+ "CG", "AT", "AC", "AA", "AG", "GT", "GC", "GA", "GG"))

[1] "TCTTTCCATTCTTCTAGTGTACCCGTATACGTGTCTGTGTACTTCAACAACCTTAT"

> seqNcbi("BY608190", file = "BY608190.fa")

[1] 1

> readFasta("BY608190.fa")
```

```

> a = matrix(c(1, 5, 15, 45, 17, 38, 100, 120, 130, 140, 135, 145,
+ 142, 160), ncol = 2, byrow = TRUE)
> b = matrix(c(15, 18, 28, 45, 1, 10, 15, 20, 25, 40, 17, 23, 35,
+ 38, 100, 105, 110, 120), ncol = 2, byrow = TRUE)
> a <- as.segSet(a)
> b <- as.segSet(b)
> A = list(matrix(c(1, 15, 17, 5, 45, 38), ncol = 2), matrix(c(100,
+ 120), ncol = 2), matrix(c(130, 135, 140, 145), ncol = 2),
+ matrix(c(142, 160), ncol = 2))
> B = list(matrix(c(15, 28, 18, 45), ncol = 2), matrix(c(1, 15,
+ 25, 10, 20, 40), ncol = 2), matrix(c(17, 35, 23, 38), ncol = 2),
+ matrix(c(100, 110, 105, 120), ncol = 2))
> A = as.globalSeg(A)
> B = as.globalSeg(B)
> c = or(a, b)
> d = and(a, b)
> e = not(a, b)
> f = Xor(a, b)
> par(mfrow = c(4, 1))
> plot(a, xlim = c(1, 160), main = "a (segSet)")
> plot(b, xlim = c(1, 160), main = "b (segSet)")
> plot(A, xlim = c(1, 160), main = "A (globalSet)")
> plot(B, xlim = c(1, 160), main = "B (globalSet)")

```

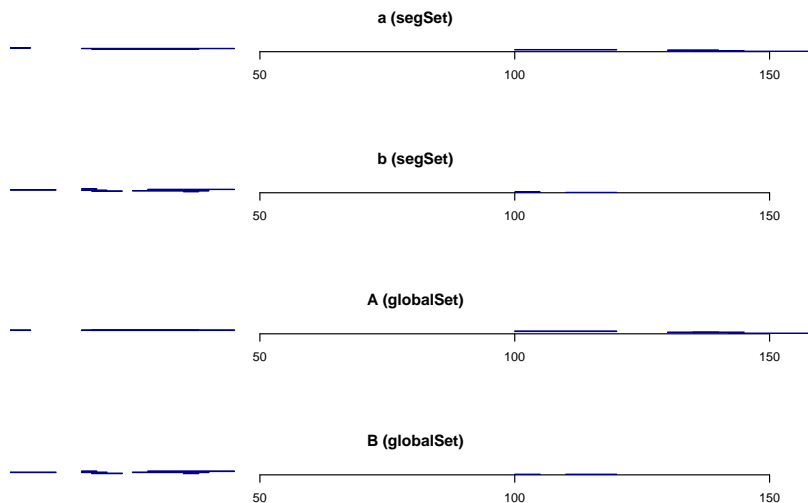


Figure 2: Representation of the two class of elements.

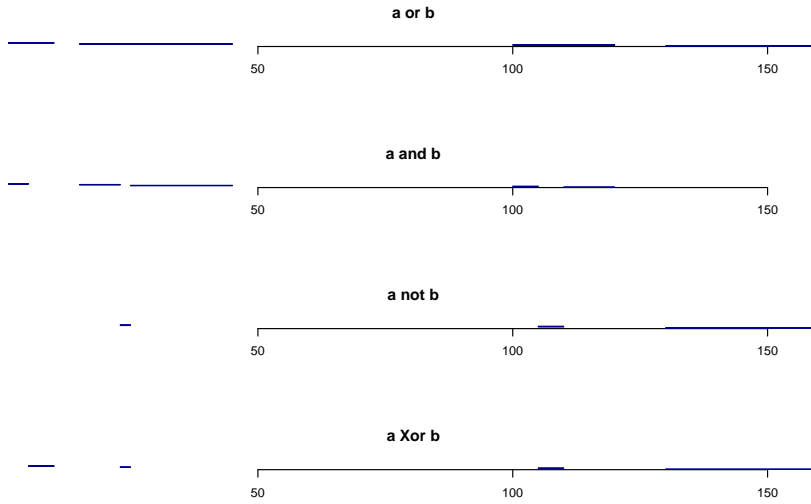


Figure 3: Union, intersection, subtraction on segments set.

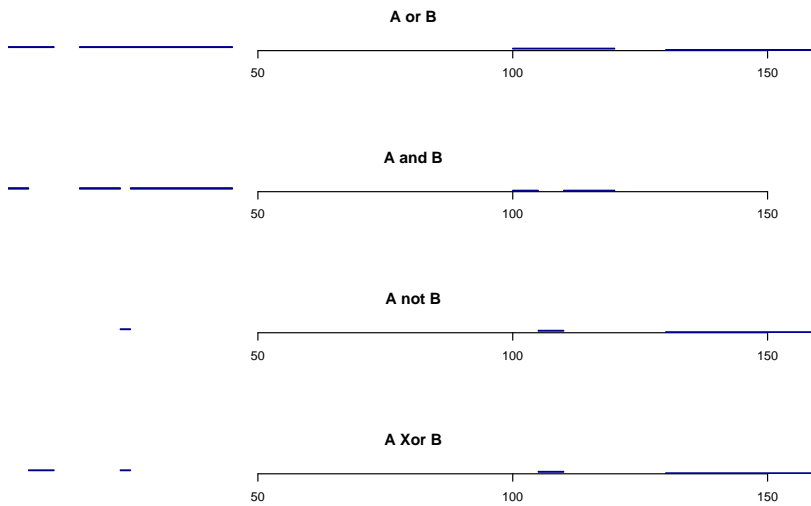


Figure 4: Union, intersection, subtraction on global segments.

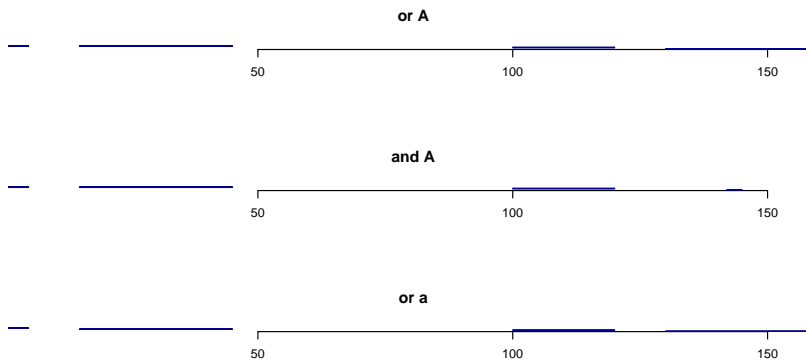


Figure 5: Unary operator.

```
[1] 0

> randomSeq(compoSeq(wsiz = 3, p = TRUE), n = 10)

[1] "TAATCTCCGTTGTXGTGCAAGTAAAAX"

> shuffleSeq(compoSeq(wsiz = 3, from = 1, to = 30, p = FALSE))

[1] "TATGTAGTTTTATGCTGTAXATTTATGTGG"
```

4.6 Profiles

4.6.1 Overview

Computes profile(s) of user defined quantities around sites of interest in sequence fragments. Profile(s) is(are) constituted of bins of equal size around the sites of interest named origins. It produces for each bin, and for each quantity the mean, the standard deviation and the number of valid events.

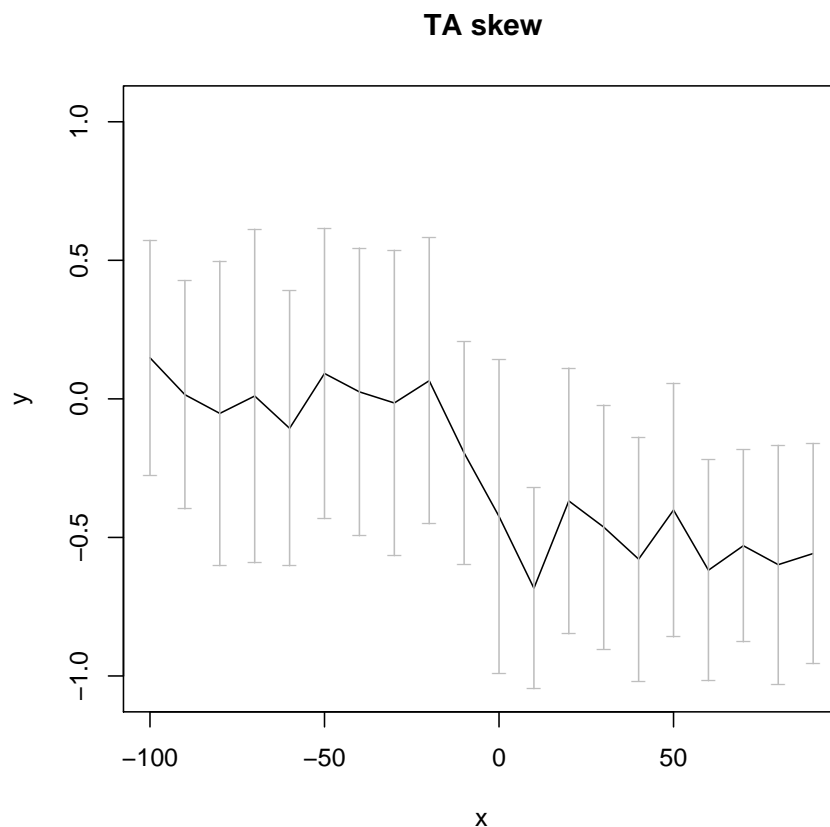
4.6.2 Examples

```
> s <- ""
> for (i in 1:20) s <- paste(s, randomSeq(n = 100), randomSeq(prob = c(0.3,
+ 1, 1, 1, 0)/3.3, n = 100), sep = "")
> placeString(s, seqno = 0)
> dens <- densityProfile(ori = 200 * (1:20) - 100, from = 200 *
+ (0:19) + 1, to = 200 * (1:20), seqno = 0, fun = seqSkew,
+ nbinsL = 10, nbinsR = 10, sizeBin = 10)
> plot(dens$skta, main = "TA skew")
> s <- ""
> for (i in 1:20) s <- paste(s, randomSeq(n = 100), randomSeq(prob = c(0.3,
+ 1, 1, 1, 0.2)/3.5, n = 100), sep = "")
> placeString(s, seqno = 1)
> dens2 <- densityProfile(ori = 200 * (1:20) - 100, from = 200 *
```

```

+ (0:19) + 1, to = 200 * (1:20), seqno = 1, fun = compoSeq,
+ nbinL = 10, nbinR = 10, sizeBin = 10)
> plot(dens2$T, main = "#T")
> dens3 <- densityProfile(ori = 200 * (1:20) - 100, from = 200 *
+ (0:19) + 1, to = 200 * (1:20), seqno = 1, fun = compoSeq,
+ nbinL = 10, nbinR = 10, sizeBin = 10, threshold = 4)
> plot(dens3$T, main = "#T")

```



5 A complete Example

In this section, we will get intronic, 3' UTR and intergenic compositions on a whole chromosome (Human: chromosome 3). Data will be retrieved from Ucsd Mysql server, (it requires to install RMySQL package).

First download a fasta file of chromosome 3:

```

download.file('http://hgdownload.cse.ucsc.edu/goldenPath/hg17/chromosomes/chr3.fa.gz',
             destfile='chr3.fa.gz')

```

Gets all information on this gene from Ucsd Mysql database:

```

library(RMySQL)
con <- dbConnect("MySQL", host="genome-mysql.cse.ucsc.edu",
                 user="genome", dbname="hg17")

```

```

rs <- dbSendQuery(con, statement = paste(
  "SELECT * ",
  "FROM knownGene",
  "WHERE chrom = 'chr3'",
  "AND strand = '+' "
))
ucscSQLdataKnownGene <- fetch(rs, n = -1)

rs <- dbSendQuery(con, statement = paste(
  "SELECT * ",
  "FROM chr3_mrna"
))
ucscSQLdataMRNA <- fetch(rs, n = -1)

```

Now we transform matrix of data (n row: one by gene) to intervals (a list of n elements, one by gene).

We will transform something like

```
ucscSQLdataMRNA[1:3,]
```

to a list adapted to our computations:

```

Genes3plus <- apply(ucscSQLdataKnownGene,1,function(u){
  matrix( c(as.integer(strsplit(as.character(u[9]),",")[[1]]),
    as.integer(strsplit(as.character(u[10]),",")[[1]])),
    ncol=2)
})
Genes3plus <- as.globalSeg(Genes3plus)

getsOneExon <- function(u)
{
  start<- as.integer(strsplit(as.character(u[22]),",")[[1]])
  stop <- as.integer(strsplit(as.character(u[20]),",")[[1]]) + start
  if(length(start) >0)
    return(matrix(c(start,stop),ncol=2))
  else
    return(NULL)
}

AllMrna3 <- apply(ucscSQLdataMRNA,1,getsOneExon)
AllMrna3 <- as.globalSeg(AllMrna3)
rm(ucscSQLdataKnownGene,ucscSQLdataMRNA,rs)

```

```
AllMrna3[1:4]
```

Here we get **Genes3plus**: an object with all known human genes on strand forward and on chromosome 3, and **AllMrna3**: an object with all possible human genes on chromosome 3.

Now we easily deduce all known introns (on strand forward), and all possible genic regions:

```

Genes3ranges <- range(Genes3plus)
possibleExons <- or(AllMrna3)
rangeDNA <- matrix(range.globalSeg(possibleExons,global=TRUE),ncol=2)

notExon <- xorRecouvr(as.matrix(possibleExons),rangeDNA)
knownIntrons <- xorRecouvr(as.matrix(Genes3plus),as.matrix(Genes3ranges))

```



```
## Safe Introns
safeIntrons <- and(knownIntrons,notExon)

rangeMRNA <- range.intervals(possibleExons)
safeIntergenic <- xorRecouvr(as.matrix(possibleExons),as.matrix(rangeMRNA))

    Now compute...

readFasta("chr3.fa")
safeIntrons <- as.matrix(safeIntrons)
safeIntergenic <- as.matrix (safeIntergenic)

composIntrons <- compoSeq(from=safeIntrons[,1],to=safeIntrons[,2])
composIntergenic <- compoSeq(from=safeIntergenic[,1],to=safeIntergenic[,2])

composIntrons
composIntergenic
```

Aknowledgments

A. Viary