The *genomeIntervals* package

Julien Gagneur

20 June 2009

1 Introduction

Genomic intervals arise in many contexts, such as genome sequence annotations (exons, introns, promoters, etc.) or experimental results of genomic studies (transcripts, ChIP-on-chip enriched regions, etc.). Often, operations over collections of genomic intervals — such as merging, overlap or non-overlap detection, or the computation of distances between intervals — are needed. The *genomeIntervals* package provides tools for this. It relies on the package *intervals*, which works with general numerical intervals, and provide wrappers for most of its functions, making them easy to use in a genomic context.

2 Genome intervals classes

We think of genomic sequences as sequences of nucleotides. These intervals are mathematically represented as intervals over the integers, $\mathbb{Z}$, with all possible types of left and right closure permitted. (See the example which follows.)

The S4 class `Genome_intervals` represents a collection of genomic intervals by extending the class `Intervals_full` from the *intervals* package. Each genome interval has a `seq_name` that represents its chromosome or, more generally, its sequence of origin. The S4 class `Genome_intervals_stranded` represents genomic intervals which are strand specific.

Below, we load and show the `Genome_intervals_stranded` object `i`, a toy example provided with the dataset `gen_ints`.

```
> library( genomeIntervals )
> data("gen_ints")
> i

Object of class Genome_intervals_stranded
7 base intervals and 0 inter-base intervals(*):
i.gene.1 chr01 + [1, 2]
i.gene.2 chr01 + (2, 5)
i.gene.3 chr02 - [11, 12)
i.gene.4 chr02 - [8, 9)
i.gene.5 chr02 - [5, 10]
i.gene.6 chr02 + [4, 12]
i.gene.7 chr03 + [2, 6)
```

```
annotation:
  seq_name inter_base strand
1    chr01      FALSE      +
2    chr01      FALSE      +
3    chr02      FALSE      -
4    chr02      FALSE      -
5    chr02      FALSE      -
6    chr02      FALSE      +
7    chr03      FALSE      +
```

`Genome_intervals` can have `rownames` (e.g., `"i.gene.1"`), which behave in the same way as `matrix` rownames: they are not mandatory and need not be unique or be supplied for all rows.

The left and right end points of each interval can be accessed and modified using standard column subsetting. Their closure status can be accessed and modified similarly, using the `closed` accessor.

```
> i[,1]

i.gene.1 i.gene.2 i.gene.3 i.gene.4 i.gene.5 i.gene.6 i.gene.7
       1        2       11        8        5        4        2

> i[,2]

i.gene.1 i.gene.2 i.gene.3 i.gene.4 i.gene.5 i.gene.6 i.gene.7
       2        5       12        9       10       12        6

> closed(i)

      [,1]  [,2]
[1,]  TRUE  TRUE
[2,] FALSE FALSE
[3,]  TRUE FALSE
[4,]  TRUE FALSE
[5,]  TRUE  TRUE
[6,]  TRUE  TRUE
[7,]  TRUE FALSE

> closed(i)[2,2] <- FALSE
```

Closure status can be adjusted quickly for all intervals in an object by supplying only two values. In this case, the two values are assumed to correspond to the left and right end points. (This is not R's standard recycling behavior, but is far more useful here.)

```
> i2 <- i
> closed(i2) <- c( TRUE, FALSE )
> i2

Object of class Genome_intervals_stranded
7 base intervals and 0 inter-base intervals(*):
i.gene.1 chr01 + [1, 2)
```

```
i.gene.2 chr01 + [2, 5)
i.gene.3 chr02 - [11, 12)
i.gene.4 chr02 - [8, 9)
i.gene.5 chr02 - [5, 10)
i.gene.6 chr02 + [4, 12)
i.gene.7 chr03 + [2, 6)

annotation:
  seq_name inter_base strand
1    chr01      FALSE      +
2    chr01      FALSE      +
3    chr02      FALSE      -
4    chr02      FALSE      -
5    chr02      FALSE      -
6    chr02      FALSE      +
7    chr03      FALSE      +
```

Sequence name and strand data can be accessed and modified with the `seq_name` and `strand` accessors:

```
> seqnames(i)

[1] chr01 chr01 chr02 chr02 chr02 chr02 chr03
Levels: chr01 chr02 chr03

> strand(i)

[1] + + - - - + +
Levels: + -

> strand(i)[2] <- "-"
```

Objects can be combined using `c`. Subsetting by row returns objects of the same class as the original object. Below we use the `Genome_intervals_stranded` object `j`, also provided with the dataset `gen_ints`.

```
> j

Object of class Genome_intervals_stranded
5 base intervals and 0 inter-base intervals(*):
j.gene.1 chr01 + [1, 2)
j.gene.2 chr01 + (5, 10]
j.gene.3 chr01 - [4, 6)
j.gene.4 chr02 + [12, 15]
j.gene.5 chr02 - [8, 9)

annotation:
  seq_name inter_base strand
1    chr01      FALSE      +
2    chr01      FALSE      +
3    chr01      FALSE      -
4    chr02      FALSE      +
5    chr02      FALSE      -
```

```
> c( i[1:3,], j[1:2,] )

Object of class Genome_intervals_stranded
5 base intervals and 0 inter-base intervals(*):
i.gene.1 chr01 + [1, 2]
i.gene.2 chr01 - (2, 5)
i.gene.3 chr02 - [11, 12)
j.gene.1 chr01 + [1, 2)
j.gene.2 chr01 + (5, 10]

annotation:
  seq_name inter_base strand
1    chr01      FALSE      +
2    chr01      FALSE      -
3    chr02      FALSE      -
4    chr01      FALSE      +
5    chr01      FALSE      +
```

The slot `annotation` is a `data.frame` that stores `seq_name`, `strand` and the `inter_base` logical vector (explained later). Additional columns may be added for extra, user-defined annotation of the intervals. Subsetting annotated objects does what it should:

```
> annotation(i)

  seq_name inter_base strand
1    chr01      FALSE      +
2    chr01      FALSE      -
3    chr02      FALSE      -
4    chr02      FALSE      -
5    chr02      FALSE      -
6    chr02      FALSE      +
7    chr03      FALSE      +

> annotation(i)$myannot = rep( c("my", "annot"), length=nrow(i) )
> annotation(i[2:3,])

  seq_name inter_base strand myannot
2    chr01      FALSE      -   annot
3    chr02      FALSE      -      my
```

Columns of the slot `annotation` can be directly accessed and modified via the `[[` and the `$` operators.

```
> i$myannot

[1] "my"    "annot" "my"    "annot" "my"    "annot" "my"

> i[["myannot"]]

[1] "my"    "annot" "my"    "annot" "my"    "annot" "my"
```

The `close_intervals` method returns a representation which is adjusted to have closed left and right end points, standardizing results. Note that `close_intervals` does not change the content of the intervals, only their representation. The companion methods `open_intervals` and `adjust_closure`, also imported from the package `intervals`, permit the other transformations.

```
> close_intervals(i)

Object of class Genome_intervals_stranded
7 base intervals and 0 inter-base intervals(*):
i.gene.1 chr01 + [1, 2]
i.gene.2 chr01 - [3, 4]
i.gene.3 chr02 - [11, 11]
i.gene.4 chr02 - [8, 8]
i.gene.5 chr02 - [5, 10]
i.gene.6 chr02 + [4, 12]
i.gene.7 chr03 + [2, 5]

annotation:
  seq_name inter_base strand myannot
1    chr01      FALSE      +      my
2    chr01      FALSE      -   annot
3    chr02      FALSE      -      my
4    chr02      FALSE      -   annot
5    chr02      FALSE      -      my
6    chr02      FALSE      +   annot
7    chr03      FALSE      +      my
```

We define the *size* of a genomic interval to be the number of bases it contains.

```
> size(i)

i.gene.1 i.gene.2 i.gene.3 i.gene.4 i.gene.5 i.gene.6 i.gene.7
       2        2        1        1        6        9        4
```

Constructing a `Genome_intervals` or a `Genome_intervals_stranded` from scratch is done by a call to `new` providing the matrix of end points, the matrix of closures (or faster a single value as shown below) and the `annotation` data frame. For `Genome_intervals_stranded`, make sure the `strand` column of the `annotation` data frame is a factor with two levels.

```
> new(
+     "Genome_intervals_stranded",
+     matrix(c(1, 2, 2, 5), ncol = 2),
+     closed = TRUE,
+     annotation = data.frame(
+             seq_name = c("chr01","chr02"),
+             inter_base = FALSE,
+             strand = factor( c("+", "+"), levels=c("+", "-") )
+     )
+ )
```

```
Object of class Genome_intervals_stranded
2 base intervals and 0 inter-base intervals(*):
chr01 + [1, 2]
chr02 + [2, 5]

annotation:
  seq_name inter_base strand
1    chr01      FALSE      +
2    chr02      FALSE      +
```

3 Overlap and set operations

3.1 Overlap

The `interval_overlap` method identifies, for each interval of the `from` object, all overlapping intervals from the `to` object. It works by `seq_name` and (if applicable) `strand`.

```
> interval_overlap( from=i, to=j )

[[1]]
[1] 1

[[2]]
[1] 3

[[3]]
integer(0)

[[4]]
[1] 5

[[5]]
[1] 5

[[6]]
[1] 4

[[7]]
integer(0)
```

3.2 Set operations

The *interval-union*, obtained by a call to `interval_union`, is defined as the union of all intervals of a `Genome_intervals` object, computed by strand and sequence name. Note that the output of `interval_union` does not include row names or any extra annotation beyond sequence name and strand, since the union process typically combines intervals, making old annotation inappropriate.

```
> interval_union(i)
```

```
Object of class Genome_intervals_stranded
5 base intervals and 0 inter-base intervals(*):
chr01 + (0, 3)
chr02 + (3, 13)
chr03 + (1, 6)
chr01 - (2, 5)
chr02 - (4, 12)

annotation:
  seq_name inter_base strand
1    chr01      FALSE      +
2    chr02      FALSE      +
3    chr03      FALSE      +
4    chr01      FALSE      -
5    chr02      FALSE      -
```

The intersection between intervals of two (or more) `Genome_intervals` objects can be obtained by using `interval_intersection`; the complement of a `Genome_intervals` object can be obtained by using `interval_complement`. (*Note:* `interval_complement` currently resorts to `-Inf` and `Inf` for outer end points. This function will be improved in a future release to utilize known chromosome extents.)

```
> interval_intersection(i,j)

Object of class Genome_intervals_stranded
4 base intervals and 0 inter-base intervals(*):
chr01 + [1, 1]
chr01 - [4, 4]
chr02 - [8, 8]
chr02 + [12, 12]

annotation:
  inter_base seq_name strand
1      FALSE    chr01      +
2      FALSE    chr01      -
3      FALSE    chr02      -
4      FALSE    chr02      +

> interval_complement(j[1:2,])

Object of class Genome_intervals_stranded
4 base intervals and 0 inter-base intervals(*):
chr01 + (-Inf, 0]
chr01 + [2, 5]
chr01 + [11, Inf)
chr02 + [-Inf, Inf]

annotation:
  seq_name inter_base strand
1    chr01      FALSE      +
```

```
2    chr01      FALSE      +
3    chr01      FALSE      +
4    chr02      FALSE      +
```

4 Distance

The function `distance_to_nearest` gives the distance from each `from` interval to the nearest `to` interval. The absolute distance is returned; signed distance (reporting whether the nearest is in 5' or 3' direction, or producing a result for both directions) is left for a future release.

```
> distance_to_nearest(i,j)
```

```
[1]  0  0  3  0  0  0 NA
```

Note that the distance to nearest of a `from` interval equals 0 if and only if at least one of the `to` intervals overlaps with it.

5 Inter-base intervals

It is sometimes useful to define positions *between* two nucleotides — to represent, for example, an insertion point or an enzyme restriction site. (The GFF format provides support for such positions.) To deal with this, we consider two types of positions along genomic sequences. The *base* positions are directly at the nucleotides; all examples shown so far in this vignette deal with base positions. The *inter-base* positions, on the other hand, fall between two consecutive nucleotides. Specifically, we define an inter-base position at i to lie between bases i and $i + 1$. We then consider two types of genomic intervals: base intervals (composed of consecutive base positions) and inter-base intervals (consecutive inter-bases).

The object `k` of the data set `gen_ints` contains both base and inter-base intervals. Inter-base intervals are indicated in the display with an asterisk. The inter-base status of a given interval can be retrieved and modified using the `inter_base` accessor function. In the next example, inter-base intervals represent two insertion points, between bases 1 and 2, and between bases 8 and 9.

```
> k
```

```
Object of class Genome_intervals_stranded
3 base intervals and 2 inter-base intervals(*):
k.site.1 chr01 + [1, 2) *
k.gene.1 chr01 + (5, 10]
k.gene.2 chr01 - [4, 6)
k.gene.3 chr02 + [12, 15]
k.site.2 chr02 - [8, 9) *

annotation:
  seq_name inter_base strand
1    chr01       TRUE      +
```

```
2    chr01      FALSE      +
3    chr01      FALSE      -
4    chr02      FALSE      +
5    chr02       TRUE      -
```

```
> inter_base(k)
```

```
[1]  TRUE FALSE FALSE FALSE  TRUE
```

```
> k[inter_base(k),]
```

```
Object of class Genome_intervals_stranded
0 base intervals and 2 inter-base intervals(*):
k.site.1 chr01 + [1, 2) *
k.site.2 chr02 - [8, 9) *

annotation:
  seq_name inter_base strand
1    chr01       TRUE      +
5    chr02       TRUE      -
```

Because size is defined as the number of bases an interval contains, size is by definition 0 for all inter-base intervals.

```
> size(k)
```

```
k.site.1 k.gene.1 k.gene.2 k.gene.3 k.site.2
       0        5        2        4        0
```

Base and inter-base intervals can overlap:

```
> interval_overlap(j,k)
```

```
[[1]]
integer(0)

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] 4

[[5]]
integer(0)
```

The distance between a base and the inter-base on either side is defined to be 0.5. Thus, distances from one base to another base, or from one inter-base to another inter-base, are integer valued; distances from a base to an inter-base, on the other hand, are half-integers. Note that overlapping intervals of any type are at distance 0 from each other.

```
> distance_to_nearest(j,k)

[1] 0.5 0.0 0.0 0.0 0.5
```

Set operations are computed for the base and the inter-base intervals independently, preserving the distinction between the two interval types:

```
> interval_union(k)

Object of class Genome_intervals_stranded
3 base intervals and 2 inter-base intervals(*):
chr01 + (5, 11)
chr01 + (0, 2) *
chr02 + (11, 16)
chr01 - (3, 6)
chr02 - (7, 9) *

annotation:
  seq_name inter_base strand
1    chr01      FALSE      +
2    chr01       TRUE      +
3    chr02      FALSE      +
4    chr01      FALSE      -
5    chr02       TRUE      -

> interval_intersection(k,j)

Object of class Genome_intervals_stranded
3 base intervals and 0 inter-base intervals(*):
chr01 + [6, 10]
chr01 - [4, 5]
chr02 + [12, 15]

annotation:
  inter_base seq_name strand
1      FALSE    chr01      +
2      FALSE    chr01      -
3      FALSE    chr02      +

> interval_complement(k[1:2,])

Object of class Genome_intervals_stranded
3 base intervals and 3 inter-base intervals(*):
chr01 + (-Inf, 5]
chr01 + [11, Inf)
chr01 + (-Inf, 0] *
chr01 + [2, Inf) *
chr02 + [-Inf, Inf]
chr02 + [-Inf, Inf] *

annotation:
```

```
  seq_name inter_base strand
1    chr01      FALSE      +
2    chr01      FALSE      +
3    chr01       TRUE      +
4    chr01       TRUE      +
5    chr02      FALSE      +
6    chr02       TRUE      +
```

6 Reading and handling GFF files

Files in the GFF3 format can be loaded by the function `readGFF3`. The companion functions `parseGffAttributes` and `getGffAttribute` provide parsing facilities for GFF attributes. To demonstrate these functions, the package *genomeIntervals* comes with a simplified GFF file derived from the yeast genome database SGD (`http://yeastgenome.org`).

```
> libPath <- installed.packages()["genomeIntervals", "LibPath"]
> filePath <- file.path(
+         libPath,
+         "genomeIntervals",
+         "example_files"
+ )
> gff <- readGff3(
+         file.path( filePath, "sgd_simple.gff" ),
+         isRightOpen=FALSE,quiet=TRUE
+ )
> idpa = getGffAttribute( gff, c( "ID", "Parent" ) )
> head(idpa)
```

```
     ID          Parent
[1,] "chrI"      NA
[2,] "TEL01L-TR" NA
[3,] "TEL01L"    NA
[4,] "TEL01L-XR" NA
[5,] "YAL069W"   NA
[6,] NA          "YAL069W"
```

7 Session information

- R version 3.3.3 (2017-03-06), `x86_64-w64-mingw32`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, utils
- Other packages: BiocGenerics 0.20.0, genomeIntervals 1.30.1, intervals 0.15.1
- Loaded via a namespace (and not attached): GenomeInfoDb 1.10.3, GenomicRanges 1.26.4, IRanges 2.8.2, RCurl 1.95-4.8, S4Vectors 0.12.2, XVector 0.14.1, bitops 1.0-6, stats4 3.3.3, tools 3.3.3, zlibbioc 1.20.0