Ranges (and Data Integration)

Martin Morgan\textsuperscript{1}
Fred Hutchinson Cancer Research Center
Seattle, WA

20 November 2013

\textsuperscript{1}mtmorgan@fhcrc.org
Introduction

Importance of range concepts: conceptually...
  ▶ Genomic data and annotation can be represented by ranges
  ▶ Biological questions reflect range-based queries

Examples
  ▶ How many reads overlap each gene?
  ▶ How many reads span splice junctions?
  ▶ Where do regulatory elements bind in ChIP-seq experiments?
  ▶ Which regulatory elements are closest to differentially expressed genes?
  ▶ What sequences are common under discovered regulatory marks?

- Initial developers: Michael Lawrence, Hervé Pagès, Patrick Aboyoun

²http://www.ploscompbiol.org/article/info%3Adoi%2F10.1371%2Fjournal.pcbi.1003118
Outline

Ranges
  IRanges
  GRanges
  Other Idioms

Data Integration

Conclusions
Ranges

What is a range?

- ‘start’ and ‘end’ coordinate vectors
- Closed interval (i.e., include end points)
- Zero-width convention
- Can be ‘named’

```r
> library(IRanges)
> eg <- IRanges(start = c(1, 10, 20),
+                end = c(4, 10, 19),
+                names = c("A", "B", "C"))
> ## bigger
> start <- floor(runif(10000, 1, 1000))
> end <- start + floor(runif(10000, 0, 100))
> ir <- IRanges(start, end)
```
‘Accessors’ and simple manipulation

Accessors

▶ start, end, width, names

‘Vector’-like behavior

▶ length, [

> length(ir)
> ir[1:4]
> ir[1:4]
> start(ir[1:4])
> ir[width(ir) > 10 & width(ir) < 20]
1. Intra-range: operate on each range independently, e.g., shift
2. Inter-range: operate on several ranges of a single instance, e.g., reduce, coverage
3. Between-range: operate on two instances, e.g., findOverlaps

See table in afternoon lab!

```r
> ir <- IRanges(start=c(7, 9, 12, 14, 22:24),
+               end=c(15, 11, 12, 18, 26, 27, 28))
> shift(ir)
> rir <- reduce(ir)
> findOverlaps(ir, rir)
```
IRangesList

- Often useful to group *IRanges* into a list, with each element of the list containing 0 or more *IRanges* instances
- Operations usually work on list element

```r
irl <- split(ir, width(ir))
> reduce(irl)
```
GRanges

Builds on IRanges, IRangesList...

- ‘seqnames’ (e.g., chromosome) and ‘strand’
- (optional) ‘seqlengths’ for genome information
- (optional) ‘mcols’ for ‘metadata’ data frame on each range

```r
> library(GenomicRanges)
> genes <- GRanges(seqnames=c("chr3R", "chrX"),
+                  ranges=IRanges(
+                  start=c(19967117, 18962306),
+                  end =c(19973212, 18962925),
+                  names=c("FBgn0039155", "FBgn0085359")),
+                  strand=c("+", "-"),
+                  seqlengths=c(chr3R=27905053L, chrX=22422827L))
> mcols(genes) <-
+                  DataFrame(EntrezId=c("42865", "2768869"),
+                  Symbol=c("kal-1", "CG34330"))
```
Coordinates and accessors

Genome coordinates

▶ 1-based
▶ ‘left-most’ – ‘start’ of ranges on the minus strand are the left-most coordinate, rather than the 5’ coordinate.

Accessors

▶ seqnames, strand, seqlengths, seqlevels and like IRanges: start, end, width, names
▶ mcols; $ for direct access to metadata

> width(genes)
> genes$Symbol
Like `IRanges`, but generally seqnames- and strand-aware

- E.g., `flank` identifies *upstream* (5’) region
- E.g., `findOverlaps` checks seqnames and strand

> `flank(genes, 1000)  ## 5' flanking range`
*List classes*

- Often useful to have a list, where all elements of the list are restricted to be of the same type – like `IRangesList`
- Support for common ‘atomic’ types (`LogicalList`, `IntegerList`, `NumericList`, `CharacterList`, …) in addition to `IRangesList`, `GRangesList`, …
- Operations on list elements usually vectorized across elements

```r
> rl <- splitAsList(1:5, c("A", "B", "A", "B", "B"))
> elementLengths(rl)
> log(rl)
```
Coverage and run-length encoding

- ‘Coverage’ as the number of ranges (or genomic ranges) overlapping positions on the positive integer number line.
- Could be represented as an integer vector, but often coverage is **sparse**
- Represent as a run-length encoding – 6 0’s followed by 2 1’s, followed by 4 2’s, etc.
- Specialized functions, e.g., `slice`
- Fast and efficient for many genomic operations

```r
> cvg <- coverage(ir)
> runLength(cvg)
> runValue(cvg)
> log(cvg)
> as.numeric(log(cvg))
> slice(cvg, lower=2)
```
Outline

Ranges
  IRanges
  GRanges
  Other Idioms

Data Integration

Conclusions
Advantages of integrated data containers

We could separately define a features × samples matrix of expression values, a data.frame describing samples, and a GRanges object describing the ranges of interest, but...

- Difficult and error prone to manipulate, e.g., subset, in a coordinated fashion.
- Different packages might follow different conventions for representing data, e.g., samples × features representation of expression values.

Instead...

- Create a class that integrates different data types
- Re-use established classes as much as possible
SummarizedExperiment

- assays: feature $\times$ sample matrices
- colData: DataFrame of sample attributes
- rowData: GRanges / GRangesList of features
- Coordination between assays, colData and rowData

```r
> library(GenomicRanges)
> ?SummarizedExperiment
> example(SummarizedExperiment)
> sset
> dim(assays(sset)[[1]])
> colData(sset)
> rowData(sset)
```
SummarizedExperiment – manipulation

- Use $ to access colData
- Use range-based operations, e.g., %over% (does the left-hand side overlap the right-hand side?) for row-based queries

```r
> sset$Treatment
> sset[, sset$Treatment == "ChIP"]
> roi <- GRanges("chr1", IRanges(1, 249250621))
> sset[sset %over% roi, ]
```
Outline

Ranges
- IRanges
- GRanges
- Other Idioms

Data Integration

Conclusions
Conclusions

Ranges

▶ Suitable for many biological questions
▶ Very rich and flexible software
▶ Performs well for large genomic data

Flexible integrated data containers

▶ Less error-prone
▶ Convenient
▶ Interoperability between packages