IRanges, GenomicRanges, and Biostrings
Bioconductor Infrastructure Packages for Sequence Analysis

Patrick Aboyoun

Fred Hutchinson Cancer Research Center

7-9 June, 2010
Outline

Introduction

Genomic Intervals with Data

Coverage and Other Piecewise Constant Measures

Long Biological Strings

Developer’s Notes

Resources
Bioconductor Sequence Packages
Bioconductor Sequence Infrastructure Packages

**IRanges**

- Long sequences (compressed & pointer referenced)
- Views on long sequences
- Integer interval tools (e.g. interval overlap)
- Genomic intervals I (*RangedData*)

**GenomicRanges**

- Genomic intervals II (*GenomicRanges*)
- Discontiguous genomic interval sets (*GenomicRangesList*)

**Biostrings**

- Long DNA/RNA/amino acids sequences
- Sequence & PWM matching and pairwise alignment tools
Bioconductor Sequence Infrastructure Classes

Ranges (as sequences & intervals)
IRanges

Genomic intervals with data
GRanges, RangedData

Genomic interval sets (e.g. spliced transcripts)
GRangesList

Long piecewise constant sequences
Rle, RleList

Long (biological) strings
DNAString, RNAString, AAString, BString, DNAStringSet, ...

Views on long sequences
RleViews, RleListViews, XStringViews, ...
Concept 1: Run-Length Encoding (RLE)

Issue

▶ Chromosomes can be hundreds of millions of base pairs long, making them hard to manage in computer memory.
▶ Fortunately, coverage vectors tend to follow an integer step function.

Solution

▶ Run-length encoding (RLE) is a common compression technique for storing long sequences with lengthy repeats.
▶ An RLE couples values with run lengths, e.g. the vector 0, 0, 0, 1, 1, 2 would be represented as (3) 0’s, (2) 1’s, and (1) 2.
▶ The IRanges package uses the Rle and RleList classes to house coverage vectors.
Concept II: Sequence Views

Issue

- Chromosomes can be hundreds of millions of base pairs long, making subsequence selection inefficient.

Solution

- Store the original sequence using a pass-by-reference semantic.
- Associate ranges with the sequence to select subsequence.
- Example:
  - 7007-letter sequence:
    
    ```
    <<SNIP-3000>>AGATTCA<<SNIP-4000>>
    ```
  - View range: [3001, 3007]
  - $\Rightarrow$ 7-letter subsequence: AGATTCA
Outline

Introduction

Genomic Intervals with Data

Coverage and Other Piecewise Constant Measures

Long Biological Strings

Developer’s Notes

Resources
Naive representation for intervals with data

Data characteristics

- Genomic coordinates consist of chromosome, position, and potentially strand information
- May have additional values, such as GC content or alignment coverage

*data.frame* approach

```r
> chr <- c("chr1", "chr2", "chr1")
> strand <- c("+", "+", "-"")
> start <- c(3L, 4L, 1L)
> end <- c(7L, 5L, 3L)
> naive <- data.frame(chr = chr, strand = strand,
+                      start = start, end = end)
```
BioC representations for intervals with data

**GRanges**

- Used by *GenomicFeatures*, a transcript annotation generator
- Intervals not required to be grouped by chromosome/contig
- Methods strand aware
- *GRangesList* class can hold exons within spliced transcripts

**RangedData**

- Used by *rtracklayer*, a genome browser interface
- Intervals grouped by chromosome/contig
- Methods strand unaware
- Preceded *GRanges* class
GRanges construction

GRanges constructor

- Instances are created using the GRanges constructor.
- Starts and ends are wrapped in an IRanges constructor.
- Chromosome/contig supplied to seqnames argument.
- Underlying sequence lengths can be supplied to seqlengths argument.

GRanges example

```r
> bioc <- GRanges(seqnames = chr,
+ ranges = IRanges(start = start, end = end),
+ strand = strand,
+ seqlengths = c("chr1" = 24, "chr2" = 18))
```
**GRanges display**

**GRanges show method**

```r
> bioc

GRanges with 3 ranges and 0 elementMetadata values

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt; &lt;Rle&gt;</td>
</tr>
<tr>
<td>[1]</td>
<td>chr1</td>
<td>[3, 7]  +</td>
</tr>
<tr>
<td>[3]</td>
<td>chr1</td>
<td>[1, 3]  -</td>
</tr>
</tbody>
</table>

seqlengths

<table>
<thead>
<tr>
<th>chr1</th>
<th>chr2</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>18</td>
</tr>
</tbody>
</table>

**Note**

- Optional interval data would appear to the right of | divider.
GRanges class decomposition

GRanges slots

```r
> getSlots("GRanges")
```

- seqnames: "Rle"
- ranges: "IRanges"
- strand: "Rle"
- seqlengths: "integer"
- elementMetadata: "ANY"
- elementType: "character"
- metadata: "list"

Notes

- If (mostly) sorted, Rle vectors reduce memory usage and provide faster group selection
- elementMetadata holds optional interval data
- metadata holds optional whole object info
Interval operations

Intra-interval
flank, resize, shift

Inter-interval I
disjoin, gaps, reduce, range

Inter-interval II
coverage

Between two interval sets I
union, intersect, setdiff

Between two interval sets II
punion, pintersect, psetdiff

Between two interval sets III
findOverlaps, countOverlaps, %in%, match

Low level
start, end, width
Creating a new *GRanges* object

New object to use in interval operations

```r
> ir <- IRanges(c(1, 8, 14, 15, 19, 34, 40),
+       width=c(12, 6, 6, 15, 6, 2, 7))
> strand <- rep(c("+", "+"), c(4,3))
> grngs <- GRanges(seqnames = "chr1", ranges = ir,
+       strand = strand,
+       seqlengths = c("chr1" = 50))
```

- blue = positive strand
- red = negative strand
**GRanges subsetting**

**seqselect**

```r
> seqselect(grngs, strand(grngs) == "-")
```

GRanges with 3 ranges and 0 elementMetadata values

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
</tr>
<tr>
<td>[1]</td>
<td>chr1</td>
<td>[19, 24]</td>
</tr>
<tr>
<td>[2]</td>
<td>chr1</td>
<td>[34, 35]</td>
</tr>
<tr>
<td>[3]</td>
<td>chr1</td>
<td>[40, 46]</td>
</tr>
</tbody>
</table>

**seqlengths**

<table>
<thead>
<tr>
<th>chr1</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
</tr>
</tbody>
</table>

**Other functions**

[, head, tail, window, subset, subsetByOverlaps]
Intra-interval (1/2)

Shifting intervals
If your interval bounds are off by 1, you can shift them.

> shift(grngs, 1)

GRanges with 7 ranges and 0 elementMetadata values

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;Rle&gt;</td>
</tr>
<tr>
<td>[1]</td>
<td>chr1 [ 2, 13]</td>
<td>+</td>
</tr>
<tr>
<td>[2]</td>
<td>chr1 [ 9, 14]</td>
<td>+</td>
</tr>
<tr>
<td>[5]</td>
<td>chr1 [20, 25]</td>
<td>-</td>
</tr>
<tr>
<td>[6]</td>
<td>chr1 [35, 36]</td>
<td>-</td>
</tr>
<tr>
<td>[7]</td>
<td>chr1 [41, 47]</td>
<td>-</td>
</tr>
</tbody>
</table>

seqlengths
chr1
50
Intra-interval (2/2)

Resizing intervals

“Growing” alignment intervals to an estimated fragment length.

> resize(grngs, 10)

GRanges with 7 ranges and 0 elementMetadata values

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr1</td>
<td>[ 1, 10]</td>
<td>+</td>
</tr>
<tr>
<td>chr1</td>
<td>[ 8, 17]</td>
<td>+</td>
</tr>
<tr>
<td>chr1</td>
<td>[14, 23]</td>
<td>+</td>
</tr>
<tr>
<td>chr1</td>
<td>[15, 24]</td>
<td>+</td>
</tr>
<tr>
<td>chr1</td>
<td>[15, 24]</td>
<td>-</td>
</tr>
<tr>
<td>chr1</td>
<td>[26, 35]</td>
<td>-</td>
</tr>
<tr>
<td>chr1</td>
<td>[37, 46]</td>
<td>-</td>
</tr>
</tbody>
</table>

seqlengths

| chr1 | 50 |
Inter-interval 1

- **reduce(grngs)**
- **gaps(grngs)**
- **disjoin(grngs)**
Overlap detection

Finding interval overlaps

findOverlap and countOverlaps produce a mapping and a tabulation of interval overlaps, respectively

```
> ol <- findOverlaps(grngs, reduce(grngs))
> as.matrix(ol)

     query subject
[1,] 1       1
[2,] 2       1
[3,] 3       1
[4,] 4       1
[5,] 5       2
[6,] 6       3
[7,] 7       4
```

```
> countOverlaps(reduce(grngs), grngs)

[1] 4 1 1 1
```
Elementwise counts of overlapping intervals

Coverage

- coverage counts number of ranges over each position
- Subset by strand to get stranded coverage

\[
> \text{cover} \leftarrow \text{coverage(grngs)}
\]
Outline

Introduction

Genomic Intervals with Data

Coverage and Other Piecewise Constant Measures

Long Biological Strings

Developer’s Notes

Resources
Piecewise constant measures

Issue restated

- The number of genomic positions in a genome is often in the billions for higher organisms, making it challenging to represent in memory.
- Some data across a genome tend to be sparse (i.e. large stretches of “no information”)

*Rle* and *RleList* classes

- Solve the set of problems for positional measures that tend to have consecutively repeating values.
- *Do not* address the more general problem of positional measures that constantly fluxuate, such as conservation scores.
Numerous Rle methods (1/2)

Numerous \textit{R} methods (2/2)

\textbf{Arith}
\begin{itemize}
  \item +, -, *, ^, \%, \%,/\%, /
\end{itemize}

\textbf{Compare}
\begin{itemize}
  \item ==, >, <, !=, <=, >=
\end{itemize}

\textbf{Logic}
\begin{itemize}
  \item &,
  \item |
\end{itemize}

\textbf{Math}
\begin{itemize}
  \item abs, sign, sqrt, ceiling, floor, trunc, cummax, cummin, cumprod, cumsum, log, log10, log2, log1p, acos, acosh, asin, asinh, ...
\end{itemize}

\textbf{Math2}
\begin{itemize}
  \item round, signif
\end{itemize}

\textbf{Summary}
\begin{itemize}
  \item max, min, range, prod, sum, any, all
\end{itemize}

\textbf{Complex}
\begin{itemize}
  \item Arg, Conj, Im, Mod, Re
\end{itemize}
Coverage example

Coverage from a *Saccharomyces cerevisiae* (Yeast) experiment contained in two objects `posCover` & `negCover`

```r
class(posCover), class(negCover))
[1] "SimpleRleList" "SimpleRleList"

posCover[["chrI"]]

'integer' Rle of length 230208 with 4814 runs
Lengths:  410  150  1470  150  269  ...  121  5  24  1252
Values:  0  1  0  1  0  ...  3  2  1  0

equivCover[["chrI"]]

'integer' Rle of length 230208 with 4641 runs
Lengths:  2267  22  128  22  456  ...  150  914  150  62
Values:  0  1  2  1  0  ...  1  0  1  0

posCover[["chrI"]]] + negCover[["chrI"]]

'integer' Rle of length 230208 with 8761 runs
Lengths:  410  150  1470  150  87  ...  126  914  150  62
Values:  0  1  0  1  0  ...  1  0  1  0
```
Plotting coverage

Custom function

```r
> plotCoverage <-
+ function(x, chrom, start=1, end=length(x[[chrom]]), col="blue",
+           xlab="Index", ylab="Coverage", main=chrom)
+ {
+   xWindow <- as.vector(window(x[[chrom]], start, end))
+   x <- start:end
+   xlim <- c(start, end)
+   ylim <- c(0, max(xWindow))
+   plot(x = start, y = 0, xlim = xlim, ylim = ylim,
+         xlab = xlab, ylab = ylab, main = main, type = "n")
+   polygon(c(start, x, end), c(0, xWindow, 0), col = col)
+ }
```
Plotting coverage on one strand

Plotting chr1+ coverage

> plotCoverage(posCover, "chrI")
Plotting stranded coverage

Custom function

```r
> plotCoverageStrands <-
+ function(pos, neg, chrom, start=1, end=length(pos[[chrom]]),
+     pos.col="blue", neg.col="red", xlab="Index",
+     ylab="Coverage", main=chrom)
+ {
+     posWindow <- as.vector(window(pos[[chrom]], start, end))
+     negWindow <- as.vector(window(neg[[chrom]], start, end))
+     x <- start:end
+     xlim <- c(start, end)
+     ylim <- c(-1, 1) * min(max(posWindow), max(negWindow))
+     plot(x = start, y = 0, xlim = xlim, ylim = ylim,
+          xlab = xlab, ylab = ylab, main = main, type = "n")
+     polygon(c(start, x, end), c(0, posWindow, 0), col = pos.col)
+     polygon(c(start, x, end), c(0, - negWindow, 0), col = neg.col)
+ }
```
Plotting coverage on both strands

Plotting chr1 coverage, both strands

> `plotCoverageStrands(posCover, negCover, "chrI")`
Plotting Coverage on both strands

Plotting chr1 coverage, both strands

\[ \text{plotCoverageStrands(posCover, negCover, "chrI", 135000, 145000)} \]
Smoothing coverage

Running window mean

```r
> posSmoothCover <- round(runmean(posCover, 75, endrule = "constant"))
> negSmoothCover <- round(runmean(negCover, 75, endrule = "constant"))
> plotCoverageStrands(posSmoothCover,negSmoothCover,"chrI",135000,145000)
```
Combining coverage

Combining coverage using "parallel" minimums

```r
> combSmoothCover <- mendoapply(pmin,
+                              posSmoothCover,
+                              negSmoothCover)
> identical(class(posSmoothCover), class(combSmoothCover))
[1] TRUE
```

- The `mendoapply` function defined in `IRanges` packages as a member of the apply family.
  - Performs elementwise operations across multiple inputs of the same type.
  - Returns an object of the same type as the inputs.
- The minimum coverage value on either strand can be computed using `pmin`. 
Plotting combined coverage

Plotting chr1, combined strands

> plotCoverage(combSmoothCover, "chr1", 135000, 145000)
Island selection

> islands <- slice(combSmoothCover, lower=1)
> islandsWithWidePeaks <- islands[viewMaxs(islands) >= 8L & 
+     width(islands) >= 500L]
> islandsWithWidePeaks

SimpleRleViewsList of length 16
$chrI
Views on a 230208-length Rle subject

views:

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>width</th>
</tr>
</thead>
<tbody>
<tr>
<td>31430</td>
<td>33109</td>
<td>1680</td>
</tr>
<tr>
<td>35071</td>
<td>36420</td>
<td>1350</td>
</tr>
<tr>
<td>42744</td>
<td>44893</td>
<td>2150</td>
</tr>
<tr>
<td>57855</td>
<td>58490</td>
<td>636</td>
</tr>
<tr>
<td>61291</td>
<td>62604</td>
<td>1314</td>
</tr>
<tr>
<td>71685</td>
<td>73511</td>
<td>1827</td>
</tr>
<tr>
<td>106221</td>
<td>108629</td>
<td>2409</td>
</tr>
<tr>
<td>139254</td>
<td>141574</td>
<td>2321</td>
</tr>
<tr>
<td>142594</td>
<td>143428</td>
<td>835</td>
</tr>
<tr>
<td>154057</td>
<td>154762</td>
<td>706</td>
</tr>
<tr>
<td>169261</td>
<td>170429</td>
<td>1169</td>
</tr>
</tbody>
</table>

<15 more elements>
Common methods for Views objects

- Subset via [, [[, etc.
- Manage edge cases via trim & restrict
- Ranges operations such as start, end, width, etc.
- Perform within view calculations via viewSums, viewMins, viewMaxs, viewWhichMins, viewWhichMaxs, viewApply
Outline

Introduction

Genomic Intervals with Data

Coverage and Other Piecewise Constant Measures

Long Biological Strings

Developer’s Notes

Resources
Long biological string framework

Biostrings string types

```r
> library(Biostrings)
> names(completeSubclasses(getClass("XString")))

[1] "BString"   "DNAString" "RNAString" "AAString"
```

DNA

```r
> data(yeastSEQCHR1)
> c(class(yeastSEQCHR1), nchar(yeastSEQCHR1))

[1] "character" "230208"

> yeast1 <- DNAString(yeastSEQCHR1)
> yeast1

230208-letter "DNAString" instance
seq: CCACACCACACCCACACACCCACACACC...GGTGTGGTGGGCTGTGGTGGGCTGTGGTGGG

> IUPAC_CODE_MAP

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>M</th>
<th>R</th>
<th>W</th>
<th>S</th>
<th>Y</th>
</tr>
</thead>
</table>
| "A"| "C"| "G"| "T"| "AC"| "AG"| "AT"| "CG"| "CT"
| K  | V  | H  | D  | B  | N  |
| "GT"| "ACG"| "ACT"| "AGT"| "CGT"| "ACGT"
```
List of strings

Biostrings string list types

> head(names(completeSubclasses(getClass("XStringSet"))), 4)
[1] "BStringSet"  "DNAStringSet"  "RNAStringSet"  "AAStringSet"

DNA strings

> data(srPhiX174)
> length(srPhiX174)
[1] 1113
> head(srPhiX174, 3)

A DNAStringSet instance of length 3

<table>
<thead>
<tr>
<th>width</th>
<th>seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>GTTATTATACCGTCAAGGACTGTGTGACTATTGAC</td>
</tr>
<tr>
<td>35</td>
<td>GGTGGTTATTATACCGTCAAGGACTGTGTGACTAT</td>
</tr>
<tr>
<td>35</td>
<td>TACCGTCAAGGACTGTGTGACTATTCGTCCTTC</td>
</tr>
</tbody>
</table>
XString class decomposition

**XString slots**

```r
> getSlots("XString")

      shared       offset       length elementMetadata
"SharedRaw"     "integer"     "integer"            "ANY"
elementType    metadata
"character"     "list"
```

```r
> getSlots("XStringSet")

      pool       ranges elementMetadata
"SharedRaw_Pool" "GroupedIRanges"            "ANY"
elementType    metadata
"character"     "list"
```

**Notes**

- shared, offset, length, pool, and ranges slots regulate pass-by-reference semantic.
- metadata slot can be used to hold annotation information.
Basic string utilities

Subsequence selection
subseq, Views

Letter frequencies
alphabetFrequency, dinucleotideFrequency,
trinucleotideFrequency, oligonucleotideFrequency,
letterFrequencyInSlidingView, uniqueLetters

Letter consensus
consensusMatrix, consensusString

Letter transformation
reverse, complement, reverseComplement, translate, chartr

I/O
read.DNAStringSet, read.RNAStringSet, read.AAStringSet,
read.BStringSet, write.XStringSet, save.XStringSet
String matching/alignment utilities

matchPDict
matchPDict, countPDict, whichPDict, vmatchPDict, vcountPDict, vwhichPDict

vmatchPattern
matchPattern, countPattern, vmatchPattern, vcountPattern, neditStartingAt, neditEndingAt, isMatchingStartingAt, isMatchingEndingAt, which.isMatchingStartingAt, which.isMatchingEndingAt

pairwiseAlignment
pairwiseAlignment, stringDist

matchPWM
matchPWM, countPWM

OTHER
matchLRPatterns, trimLRPatterns, matchProbePair, findPalindromes, findComplementedPalindromes
Letter frequencies

Single-letter frequencies

> alphabetFrequency(yeast1, baseOnly=TRUE)

A   C   G   T   other
69830 44643 45765 69970   0

Multi-letter frequencies

> dinucleotideFrequency(yeast1)

AA   AC   AG   AT   CA   CC   CG   CT   GA   GC
23947 12493 13621 19769 15224 9218 7089 13112 14478 8910
GG   GT   TA   TC   TG   TT
9438 12938 16181 14021 15617 24151

> head(trinucleotideFrequency(yeast1), 12)

AAA   AAC   AAG   AAT   ACA   ACC   ACG   ACT   AGA   AGC   AGG   AGT
8576  4105  4960  6306  3924  2849  2186  3534  4537  2680  2707  3697
Basic transformations

Standard transformations

> x
  21-letter "DNAString" instance
  seq: TCAACGTGAATAGCCTACCG
> reverseComplement(x)
  21-letter "DNAString" instance
  seq: CGGTACGCTATTCAACGTTGA
> translate(x)
  7-letter "AAString" instance
  seq: STLNSVP

Bisulfite transformation

> library(BSgenome.Celegans.UCSC.ce2)
> alphabetFrequency(Celegans$chrII, baseOnly=TRUE)
  A   C   G   T   other
  4878194 2769208 2762193 4869710 3
> chrIIbis <- chartr("C", "T", Celegans$chrII)
> alphabetFrequency(chrIIbis, baseOnly=TRUE)
  A   C   G   T   other
  4878194    0 2762193 7638918 3
Letter consensus

Consensus matrix

```r
> snippet <- subseq(head(sort(srPhiX174), 5), 1, 10)
> consensusMatrix(snippet, baseOnly=TRUE)

A  5  5  1  0  4  2  1  0  1  0
C  0  0  1  0  0  2  0  0  0  0
G  0  0  3  4  0  0  4  0  3
T  0  0  0  1  1  1  4  1  4  2
other 0  0  0  0  0  0  0  0  0  0
```

Consensus string

```r
> consensusString(snippet)

[1] "AAGGAMTGTK"

> consensusString(snippet, ambiguityMap = "N", threshold = 0.5)

[1] "AAGGANTGTG"
```
String matching

Match counting

```r
> data(phiX174Phage)
> genome <- phiX174Phage[["NEB03"]]
> negPhiX174 <- reverseComplement(srPhiX174)
> posCounts <- countPDict(PDict(srPhiX174), genome)
> negCounts <- countPDict(PDict(negPhiX174), genome)
> table(posCounts, negCounts)

    negCounts
posCounts     0  1030
       0      1  83
```

Match locations

```r
> matchPDict(PDict(srPhiX174[posCounts > 0]), genome)
MIndex object of length 83
```
Pairwise alignments

Alignment scores

```r
> data(phiX174Phage)
> posScore <- pairwiseAlignment(srPhiX174, genome,
+    type = "global-local", scoreOnly = TRUE)
> negScore <- pairwiseAlignment(negPhiX174, genome,
+    type = "global-local", scoreOnly = TRUE)
> cutoff <- max(pmin.int(posScore, negScore))
```

Alignments

```r
> pairwiseAlignment(srPhiX174[posScore > cutoff], genome,
+    type = "global-local")
```

Global-Local PairwiseAlignedFixedSubject (1 of 1112)

pattern:     [1] GTTATTATACCGTCAAGGACTGTGTGACTATTGAC
subject:     [2750] GTTATTATACCGTCAAGGACTGTGTGACTATTGAC
score:       69.36144
Outline

Introduction

Genomic Intervals with Data

Coverage and Other Piecewise Constant Measures

Long Biological Strings

Developer’s Notes

Resources
Long compressed sequence classes (*IRanges*)

**Rle**
- Compressed atomic vectors
- Methods for standard *R* atomic vector functions
- Concrete class with sub-typing at the slot level

**CompressedList**
- Compressed list of S4 objects
- Methods for standard *R* list functions
- Virtual class with sub-typing at the subclass level

**IRanges (as Sequences)**
- `as.integer` coercion
- Subscripting via `seqselect`, `window`, and `[`
**XVector (IRanges)**

- External pointer-based atomic vectors
- Virtual class
- Concrete subclasses:
  - *XRaw* – Underlies *Biostrings* infrastructure
  - *XInteger* – Experimental integer vector class
  - *XDouble* – Experimental real number vector class

**XString (Biostrings)**

- Virtual class
- Concrete subclasses:
  - *BString* – Any “biological” sequence
  - *DNAString* – DNA sequence
  - *RNAString* – RNA sequence
  - *AAString* – Amino acid sequence
Outline

Introduction

Genomic Intervals with Data

Coverage and Other Piecewise Constant Measures

Long Biological Strings

Developer’s Notes

Resources
Resources

**Bioconductor Web site**
- [http://bioconductor.org](http://bioconductor.org)

**Help in R**
- `help.start()` to view a help browser.
- `help(package = "Biostrings")`
- `?findOverlaps`
- `browseVignettes("GenomicRanges")`