

Representing sequencing data in R and Bioconductor

Mark Dunning, Bioinformatics Training Coordinator

Cancer Research UK
Cambridge Research Institute
Robinson Way
Cambridge

1 June 2015

Outline

Aims

Introduction

- Biostrings

- Representing the genome

Hands-on

Representing sequencing reads

Hands-on

Ranges

- IRanges

Hands-on

Dealing with aligned reads

Aims

By the end of this lecture and practical you should be familiar with

- ▶ How DNA sequences are represented in R
- ▶ How to create and compare genomic intervals
- ▶ How to read fastq and bam files into R
- ▶ Interactions between the packages

Introduction

Sequencing produces millions of reads. e.g in fastq format

Read 1

```
TGAAAGCACTTTGGAAAGTGTAACAAATCATAGAGTTGTGAGTGATTGTTATTATTATAATTATTATT
```

Read 2

```
TTGGAAAGTGTAACAAATCATAGAGTTGTGAGTGATTGTTATTATTATAATTATTATTTTGTGGCAAA
```

Read 3

```
GTTGGAAGAGACTGGGTTTACTATAAAATGAGAATAAAAGTACCTAAGTTGGAGTATTATTGGACAAC
```

Read 4

```
GAAGAGACTGGGTTTACTATAAAATGAGAATAAAAGTACCTAAGTTGGAGTATTATTGGACAACACTAGA
```

Read 5

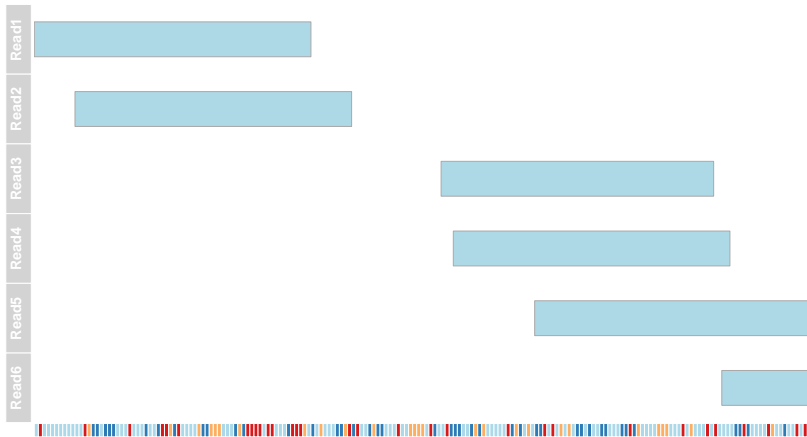
```
AAAATGAGAATAAAAGTACCTAAGTTGGAGTATTATTGGACAACACTAGAAGAGAATGTATATACATAAT
```

Read 6

```
GAAGAGAATGTATATACATAATAGACACTCAAAAAAGGGAAGCAGTAGTCTTCGTGGCTATTATTATT
```

These need to be compared to the genome (aligned) and we record the chromosome and coordinates that each sequence aligns to, often with quality information.

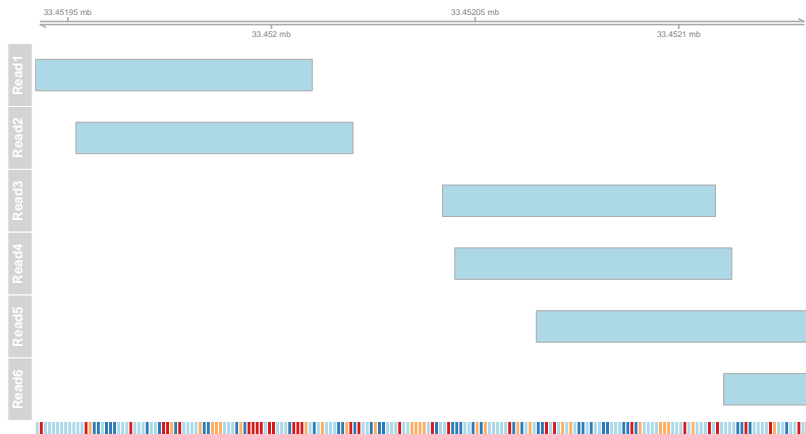
Need consistent representation of (1) genome and (2) reads



Reads come with quality score and IDs that also need to be captured

Associating reads with positions

Often we are given the mapped location of reads



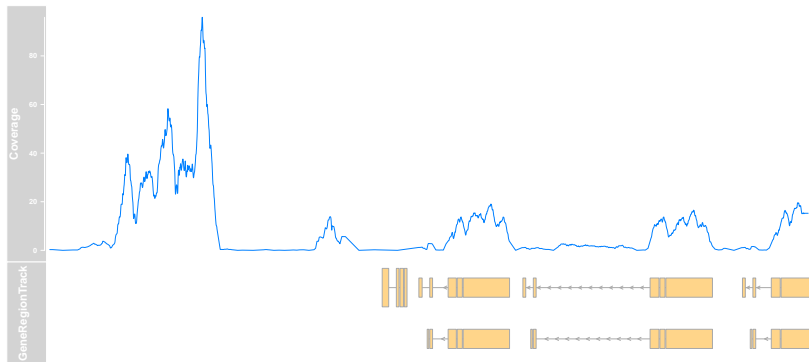
Need a way of representing alignments and associated qualities

Associating with Genomic Features

We will often want to find information about the genomic region around the reads



Or use definitions of genomic regions to interrogate the data



Need representation of genomic regions of interest

Why bother doing these things in R?

- ▶ Interactivity and data exploration
- ▶ Quality assessment
- ▶ Access to existing statistical and visualisation techniques (e.g. limma)
- ▶ Reproducibility

We will not do alignment of reads in R

Packages we will meet

- ▶ Biostrings - Manipulation of DNA sequences in R
- ▶ ShortRead - Input / output of fastq files and quality assessment
- ▶ IRanges - Low-level classes and functions for dealing with intervals of consecutive values
- ▶ GRanges - Functions for representing ranges with sequence and strand information
- ▶ Rsamtools - Input of bam files

DNA Sequences

We can represent sequences of A, T, C, G. Several useful operations are possible

```
myseq

## [1] "AGTCTGCTCCAG" "CGCAGTCGCGG"

gsub("A", "X", myseq)

## [1] "XGTCTGCTCCXG" "CGCXGTCGCGG"

nchar(myseq)

## [1] 12 11

substr(myseq, 1,3)

## [1] "AGT" "CGC"
```

Biostrings package

However, the Biostrings package is specifically-designed for biological sequences

```
library(Biostrings)
myseq <- DNASTringSet(randomStrings)
```

Biostrings operations

```
myseq
```

```
##      A DNAStringSet instance of length 100
##           width seq
##      [1]      12 AGTCTGCTCCAG
##      [2]      11 CGCAGTCGCGG
##      [3]      18 TGGTCTTTGTTCACTCTT
##      [4]      15 AGAAAAAGCCCTTCG
##      [5]      17 GTTAAGATGCTTACTGA
##      ...      ...
##      [96]      13 ACTTCCTTTTCTG
##      [97]      12 TAATGTCAAGAG
##      [98]      10 TGA CTCTCAA
##      [99]      14 TTATAGACTCTGGA
##     [100]      13 GATCACAGCGCGG
```

Biostrings operations

```
myseq[1:5]

##      A DNAStringSet instance of length 5
##      width seq
## [1]      12 AGTCTGCTCCAG
## [2]      11 CGCAGTCGCGG
## [3]      18 TGGTCTTTGTTCACTCTT
## [4]      15 AGAAAAAGCCCTTCG
## [5]      17 GTTAAGATGCTTACTGA
```

This doesn't work!

```
myseq[,1:5]
```

```
subseq(myseq, 1, 5)
```

```
##      A DNAStringSet instance of length 100
##           width seq
##      [1]      5 AGTCT
##      [2]      5 CGCAG
##      [3]      5 TGGTC
##      [4]      5 AGAAA
##      [5]      5 GTTAA
##      ...      ... ...
##     [96]      5 ACTTC
##     [97]      5 TAATG
##     [98]      5 TGACT
##     [99]      5 TTATA
##    [100]      5 GATCA
```

Similar to substr

Biostrings operations

Accessor functions must be used to retrieve the data

```
width(myseq)
```

```
##      [1] 12 11 18 15 17 12 16 16 15 11 16
##     [12] 10 11 20 20 12 10 19 10 18 15 10
##     [23] 11 19 20 17 12 10 15 19 17 16 18
##     [34] 18 13 12 13 11 20 11 15 14 10 17
##     [45] 13 19 14 19 19 14 10 14 15 11 10
##     [56] 18 20 13 19 14 19 15 11 18 10 20
##     [67] 11 14 16 10 13 13 11 14 11 10 18
##     [78] 17 19 10 18 19 10 19 10 13 11 16
##     [89] 19 12 19 11 19 17 19 13 12 10 14
##    [100] 13
```

```
table(width(myseq))
```

```
##
## 10 11 12 13 14 15 16 17 18 19 20
## 15 13  7  9  8  7  6  6  8 15  6
```


Can subset based on properties of the set

```
myseq[width(myseq)>19]
```

```
## A DNAStringSet instance of length 6
## width seq
## [1] 20 ATAGCCAGTGTTATGTACT
## [2] 20 CAGGTTGCAATTCATTGGCA
## [3] 20 ACACATGTGTCCTTCTTAAG
## [4] 20 ATGTCGATGAACGTATGGTC
## [5] 20 TTTAGGAAGCAGATGTTCTA
## [6] 20 CCCTCTCGGCAGAACGAGGG
```

```
myseq[as.character(substr(myseq,1,3))=="TTC"]
```

```
## A DNAStringSet instance of length 1
## width seq
## [1] 12 TTCCAGGGTTAC
```

Some useful string operation functions are provided

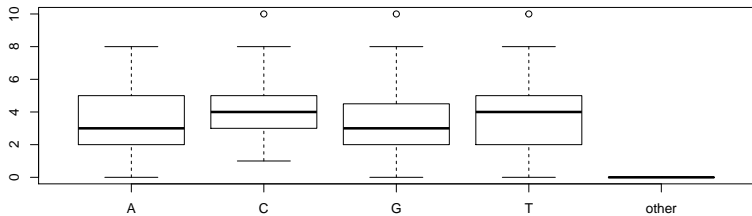
```
alphabetFrequency(myseq[1:4,], baseOnly=TRUE)
```

```
##      A C G  T other
## [1,] 2 4 3  3      0
## [2,] 1 4 5  1      0
## [3,] 1 4 3 10      0
## [4,] 6 4 3  2      0
```

```
af <- alphabetFrequency(myseq, baseOnly=TRUE)
myseq[af[,1] ==0,]
```

```
##      A DNAStringSet instance of length 3
##      width seq
## [1]      10 CTCCTGGTCC
## [2]      11 TCGCCGCCCT
## [3]      19 CGGGTGCTCGCTTCCGCCT
```

```
boxplot(af)
```



More-specialised features

```
myseq[1:2,]
```

```
##      A DNAStringSet instance of length 2
##      width seq
## [1]      12 AGTCTGCTCCAG
## [2]      11 CGCAGTCGCGG
```

```
reverse(myseq[1:2,])
```

```
##      A DNAStringSet instance of length 2
##      width seq
## [1]      12 GACCTCGTCTGA
## [2]      11 GGCGCTGACGC
```

```
reverseComplement(myseq[1:2,])
```

```
##      A DNAStringSet instance of length 2
##      width seq
## [1]      12 CTGGAGCAGACT
## [2]      11 CCGCGACTGCG
```

```
translate(myseq[1:4,])
```

```
##      A AAStringSet instance of length 4
##      width seq
## [1]      4 SLLQ
## [2]      3 RSR
## [3]      6 WSLFTL
## [4]      5 RKSPS
```

The genome as a string - BSgenome

```
library(BSgenome)
head(available.genomes())

## [1] "BSgenome.Alyrata.JGI.v1"
## [2] "BSgenome.Amelliifera.BeeBase.assembly4"
## [3] "BSgenome.Amelliifera.UCSC.apiMel2"
## [4] "BSgenome.Amelliifera.UCSC.apiMel2.masked"
## [5] "BSgenome.Athaliana.TAIR.04232008"
## [6] "BSgenome.Athaliana.TAIR.TAIR9"
```

Various versions of the human genome

```
## [1] "BSgenome.Hsapiens.NCBI.GRCh38"
## [2] "BSgenome.Hsapiens.UCSC.hg17"
## [3] "BSgenome.Hsapiens.UCSC.hg17.masked"
## [4] "BSgenome.Hsapiens.UCSC.hg18"
## [5] "BSgenome.Hsapiens.UCSC.hg18.masked"
## [6] "BSgenome.Hsapiens.UCSC.hg19"
## [7] "BSgenome.Hsapiens.UCSC.hg19.masked"
## [8] "BSgenome.Hsapiens.UCSC.hg38"
## [9] "BSgenome.Hsapiens.UCSC.hg38.masked"
```

The human genome

```
library(BSgenome.Hsapiens.UCSC.hg19)
hg19 <- BSgenome.Hsapiens.UCSC.hg19::Hsapiens
hg19

## Human genome:
## # organism: Homo sapiens (Human)
## # provider: UCSC
## # provider version: hg19
## # release date: Feb. 2009
## # release name: Genome Reference Consortium GRCh37
## # 93 sequences:
## #   chr1
## #   chr2
## #   chr3
## #   chr4
## #   chr5
## #   ...
## #   chrUn_g1000245
## #   chrUn_g1000246
## #   chrUn_g1000247
## #   chrUn_g1000248
## #   chrUn_g1000249
```

Retrieve Sequences

```
tp53 <- getSeq(hg19, "chr17", 7577851, 7590863)
```

```
tp53
```

```
## 13013-letter "DNAString" instance
```

```
## seq: TTGTATTTTTCAGTAG...GGGAAAACCCCAATC
```

```
as.character(tp53)
```

```
## [1] "TTGTATTTTTCAGTAGAGACGGGGTTTCACCGTTAGCCAGGATGGTCTCGATCTCCCAACCTC
```

```
alphabetFrequency(tp53, baseOnly=TRUE)
```

```
##      A      C      G      T other
```

```
## 3102 3375 3025 3511      0
```

```
subseq(tp53, 1000, 1010)
```

```
## 11-letter "DNAString" instance
```

```
## seq: TATAGGTGTGC
```


Timings

Don't need to load the whole genome into memory, so reading a particular sequence is **fast**

```
system.time(tp53 <- getSeq(hg19, "chr17", 7577851, 7598063))
```

```
##      user  system elapsed  
##    0.123    0.014    0.135
```

Manipulating sequences

We can now use Biostrings operations to manipulate the sequence

```
translate(subseq(tp53, 1000,1010))  
  
##    3-letter "AAString" instance  
## seq: YRC  
  
reverseComplement(subseq(tp53, 1000,2000))  
  
##    1001-letter "DNAString" instance  
## seq: CCTATGGAAACTGTGA...GTGGTGCACACCTATA
```

Later, we will show how the sequences for genomic features can be extracted

Intermission

Work through section 2 of the practical

Fastq Recap

Recall that sequence reads are represented in text format

```
readLines(path.to.my.fastq ,n=10)
```

```
## [1] "@SRR076417.586678/1"
## [2] "GAAATTAGCAGAAACCGCCTGCGACCCTTTAGTTTCTTGCTACTGGGAACCCACGTGCATG
## [3] "+"
## [4] "S=>><>;?>=?>@?=78@?@??@8?>@@@??=A=@?@@?A@@>>@?AA?@?@@?>8=?>=@
## [5] "@SRR076258.517296/1"
## [6] "TTATTGATCTTTTGTGACATGCACGTGGGTTCCCAGTAGCAAGAACTAAAGGGTCGCAGGC
## [7] "+"
## [8] "R>:====<>>>>?<?9>==@@>;9>?@@?>@@@?<=A?>@A@@@>@=@@=@@<=9@>@A?
## [9] "@SRR076944.334021/1"
## [10] "AACTAAAGGGTCGCAGGCCGGTTTCTGCTAATTTCTTTAATTCCAAGACAGTCTCAAATATT
```

It should be possible to represent these as **Biostrings** objects

The Short Read package

Has convenient functions for reading fastq files and performing quality assessment

```
library(ShortRead)
fq <- readFastq(path.to.my.fastq)
fq

## class: ShortReadQ
## length: 1000000 reads; width: 68 76 cycles
```

```
sread(fq)[1:3,]
```

```
##      A DNAStringSet instance of length 3
##      width seq
## [1]      68 GAAATTAGCAGAAA...GTGCATGTCACAA
## [2]      68 TTATTGATCTTTTG...CGCAGGCCGTTT
## [3]      68 AACTAAAGGGTCGC...AAATATTTTCTTA
```

```
quality(fq)[1:3,]
```

```
## class: FastqQuality
## quality:
##      A BStringSet instance of length 3
##      width seq
## [1]      68 S=>><>;?>=?>@?...8=?>=@:>>=>P
## [2]      68 R>:====<>>>>?...=9@>@A??8?==P
## [3]      68 F<;>;=<?:9>98...>?><6====>=>H
```

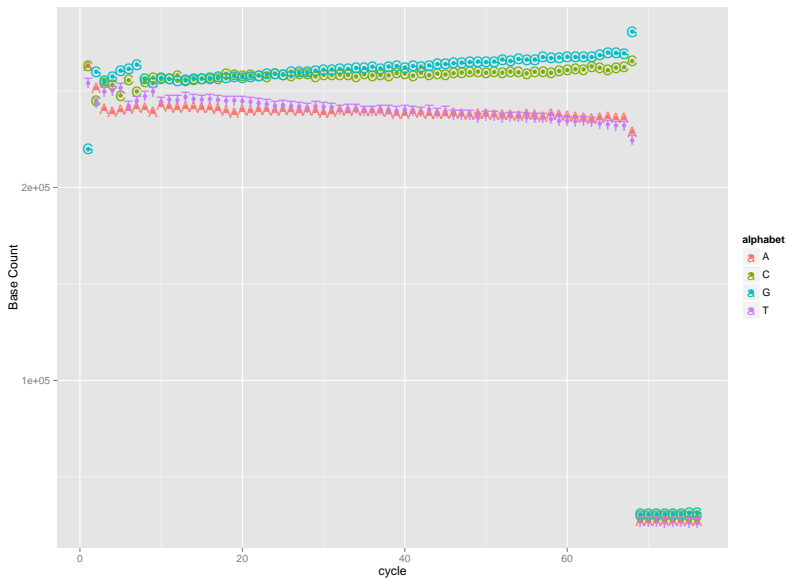
```
id(fq)[1:3]
```

```
##      A BStringSet instance of length 3  
##      width seq  
## [1]      18 SRR076417.586678/1  
## [2]      18 SRR076258.517296/1  
## [3]      18 SRR076944.334021/1
```

Could parse the ID for run names, lanes, tiles etc

```
abc <- alphabetByCycle(sread(fq))
abc[1:4, 1:8]
```

```
##           cycle
## alphabet  [,1]  [,2]  [,3]  [,4]
##           A 263154 251453 240710 239106
##           C 262943 244725 254195 253128
##           G 219843 260012 255336 257532
##           T 254060 243810 249759 250234
##           cycle
## alphabet  [,5]  [,6]  [,7]  [,8]
##           A 240026 240715 241648 241353
##           C 247750 255613 249749 254996
##           G 260568 261494 263695 256382
##           T 251656 242178 244908 247269
```

Conversion of qualities

Phred quality scores are integers from 0 to 50 that are stored as ASCII characters after adding 33. The basic R functions `rawToChar` and `charToRaw` can be used to convert

```
phred <- 1:9
phreda <- paste(sapply(as.raw((phred)+33), rawToChar), collapse=""); phreda

## [1] "\"#$%&'()*\"

as.integer(charToRaw(phreda))-33

## [1] 1 2 3 4 5 6 7 8 9

round(10^((-as.integer(charToRaw(phreda))-33)/10),2)

## [1] 0.79 0.63 0.50 0.40 0.32 0.25 0.20
## [8] 0.16 0.13
```

```
## class: FastqQuality
## quality:
##   A BStringSet instance of length 1
##   width seq
## [1]      68 S=>><>;?>=?>@?...8=?>=@:>>=P
```

```
## [1] 29 29 29 29 29 29 29 29 29 29 29 29
## [13] 29 29 29 29 29 29 29 29 29 27 29 29
## [25] 29 29 29 10 29 10 25 19 23 29 27
```

A shortcut

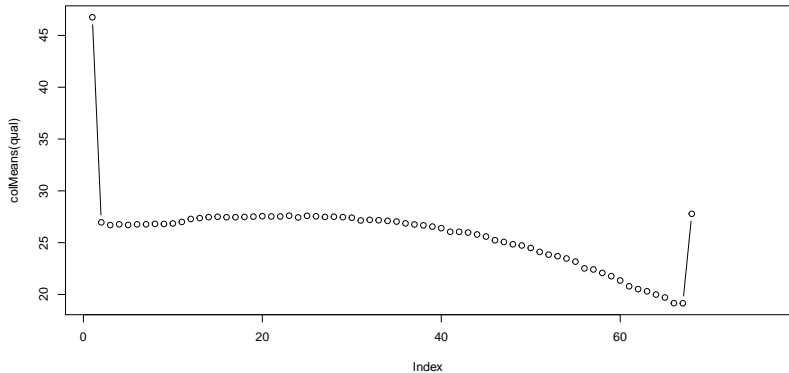
```
qual <- as(quality(fq), "matrix")  
dim(qual)
```

```
## [1] 1000000      76
```

```
qual[1,]
```

```
## [1] 50 28 29 29 27 29 26 30 29 28 30 29  
## [13] 31 30 28 30 23 31 30 31 30 30 31 23  
## [25] 30 29 31 31 31 30 30 28 32 28 31 30  
## [37] 31 31 30 32 31 31 29 29 31 30 32 32  
## [49] 30 31 30 31 31 30 29 23 28 30 28 29  
## [61] 28 31 25 29 29 28 29 47 NA NA NA NA  
## [73] NA NA NA NA
```

```
plot(colMeans(qual), type="b")
```



Read Occurrence

```
tbl <- tables(fq)
names(tbl)
```

```
## [1] "top"          "distribution"
```

```
tbl$top[1:5]
```

```
## AGGGGGAGCGGCAAGAGCAACGTGGGCACTTCTGGAGACCACGACGATTCTGCTATGAAGACACTCAG
## 17
## TATGAAGACACTCAGGAGCAAGATGGGCAAGTGGTGCTGCCACTGCTTCCCCTGCTGCAGGGGGAGCG
## 16
## GTGGGCACTTCTGGAGACCACGACGATTCTGCTATGAAGACACTCAGGAGCAAGATGGGCAAGTGGTG
## 15
## AACGTGGGCACTTCTGGAGACCACGACGATTCTGCTATGAAGACACTCAGGAGCAAGATGGGCAAGTG
## 14
## AGCGGCAAGAGCAACGTGGGCACTTCTGGAGACCACGACGATTCTGCTATGAAGACACTCAGGAGCAA
## 13
```

```
head(tbl$distribution)
```

```
##      nOccurrences nReads  
## 1                1 906589  
## 2                2  36417  
## 3                3   4773  
## 4                4    959  
## 5                5    269  
## 6                6     73
```

977827 sequences appear only once, 6801 appear twice, etc.

We can trim the reads if required

```
subseq(sread(fq), 1, 10)

##      A DNAStringSet instance of length 1000000
##              width seq
##      [1]      10 GAAATTAGCA
##      [2]      10 TTATTGATCT
##      [3]      10 AACTAAAGGG
##      [4]      10 TAATTTAGTA
##      [5]      10 CTTTTTAAGA
##      ...      ...  ...
## [999996]      10 GTAGTCTCAC
## [999997]      10 ATGTGGACCT
## [999998]      10 TGTAGTCTCA
## [999999]      10 CCCTATCCCT
## [1000000]      10 GTGTAGTCTC
```

or search for adaptor sequence

```
grep(myAdaptor, sread(fq))
```

And write the resulting files

```
write.XStringSet(...)
```


Intermission

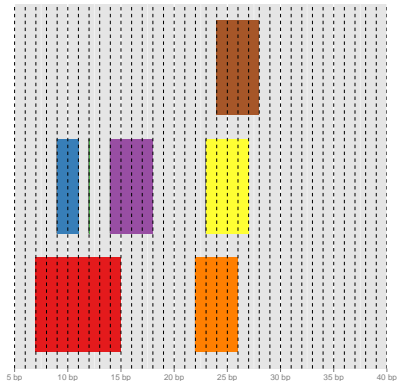
Work through section 3 of the practical

IRanges

- ▶ Genome is typically represented as linear sequence
- ▶ Ranges are an ordered set of consecutive integers defined by a start and end position
- ▶ $\text{start} \leq \text{end}$
- ▶ Ranges are a common scaffold for many genomic analyses
- ▶ Ranges can be associated with genomic information (e.g. gene name) or data derived from analysis (e.g. counts)

Suppose we want to capture information on the following intervals

Start	End
7	15
9	11
12	12
14	18
22	26
23	27
24	28



```
library(IRanges)
ir <- IRanges(
  start = c(7,9,12,14,22:24),
  end=c(15,11,12,18,26,27,28))
ir
```

```
## IRanges of length 7
##      start end width
## [1]      7  15     9
## [2]      9  11     3
## [3]     12  12     1
## [4]     14  18     5
## [5]     22  26     5
## [6]     23  27     5
## [7]     24  28     5
```

```
start(ir)
```

```
## [1] 7 9 12 14 22 23 24
```

```
end(ir)
```

```
## [1] 15 11 12 18 26 27 28
```

```
width(ir)
```

```
## [1] 9 3 1 5 5 5 5
```

Ranges as vectors

```
ir

## IRanges of length 7
##      start end width
## [1]      7  15     9
## [2]      9  11     3
## [3]     12  12     1
## [4]     14  18     5
## [5]     22  26     5
## [6]     23  27     5
## [7]     24  28     5
```

```
ir[1:2]

## IRanges of length 2
##      start end width
## [1]      7  15     9
## [2]      9  11     3

ir[width(ir)==5]

## IRanges of length 4
##      start end width
## [1]     14  18     5
## [2]     22  26     5
## [3]     23  27     5
## [4]     24  28     5
```

Common Operations

- ▶ Intra-range
 - ▶ shift - move ranges by specified amount
 - ▶ resize - change width, anchoring start, end or mid flank - Regions adjacent to start or end
 - ▶ flank - flanking regions
- ▶ Inter-range
 - ▶ reduce
 - ▶ gaps

See GRanges paper [Software for Computing and Annotating Genomic Ranges](#). (2013) PLoS Computational Biology

Shifting

We could do this the long way

```
ir2 <- IRanges(start(ir) + 5, end(ir) + 5)
```

But a shortcut is provided by IRanges

```
identical(ir2, shift(ir, 5))
```

```
## [1] TRUE
```


Shifting

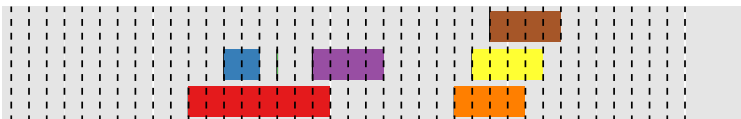
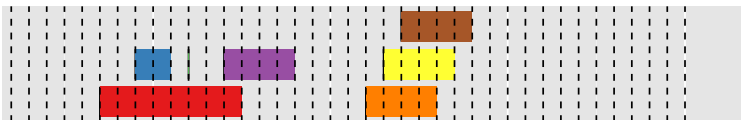
e.g. sliding windows

```
ir
```

```
## IRanges of length 7
##      start end width
## [1]      7 15      9
## [2]      9 11      3
## [3]     12 12      1
## [4]     14 18      5
## [5]     22 26      5
## [6]     23 27      5
## [7]     24 28      5
```

```
shift(ir, 5)
```

```
## IRanges of length 7
##      start end width
## [1]     12 20      9
## [2]     14 16      3
## [3]     17 17      1
## [4]     19 23      5
## [5]     27 31      5
## [6]     28 32      5
## [7]     29 33      5
```



Shifting

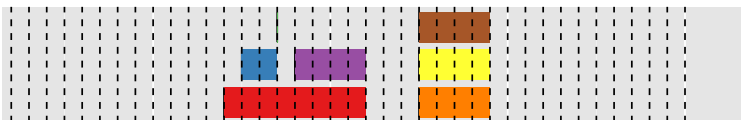
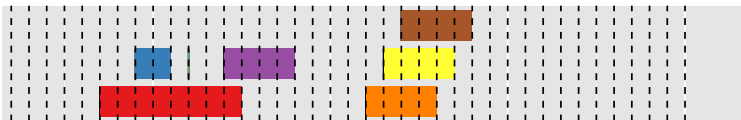
Size of shift doesn't need to be constant

```
ir
```

```
## IRanges of length 7
##      start end width
## [1]      7  15      9
## [2]      9  11      3
## [3]     12  12      1
## [4]     14  18      5
## [5]     22  26      5
## [6]     23  27      5
## [7]     24  28      5
```

```
shift(ir, 7:1)
```

```
## IRanges of length 7
##      start end width
## [1]     14  22      9
## [2]     15  17      3
## [3]     17  17      1
## [4]     18  22      5
## [5]     25  29      5
## [6]     25  29      5
## [7]     25  29      5
```



Resize

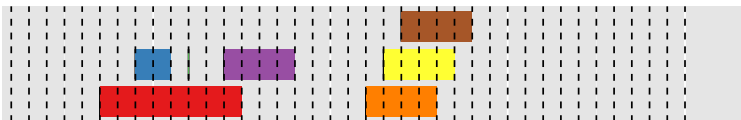
e.g. trimming reads

```
ir
```

```
## IRanges of length 7
##      start end width
## [1]      7 15      9
## [2]      9 11      3
## [3]     12 12      1
## [4]     14 18      5
## [5]     22 26      5
## [6]     23 27      5
## [7]     24 28      5
```

```
resize(ir,3)
```

```
## IRanges of length 3
##      start end width
## [1]      7  9      3
## [2]      9 11      3
## [3]     12 14      3
## [4]     14 16      3
## [5]     22 24      3
## [6]     23 25      3
## [7]     24 26      3
```



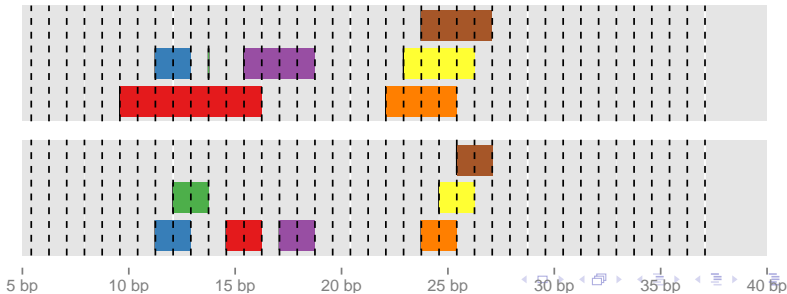
Resize

```
ir
```

```
## IRanges of length 7
##      start end width
## [1]      7  15      9
## [2]      9  11      3
## [3]     12  12      1
## [4]     14  18      5
## [5]     22  26      5
## [6]     23  27      5
## [7]     24  28      5
```

```
resize(ir,3,fix="end")
```

```
## IRanges of length 7
##      start end width
## [1]     13  15      3
## [2]      9  11      3
## [3]     10  12      3
## [4]     16  18      3
## [5]     24  26      3
## [6]     25  27      3
## [7]     26  28      3
```



Coverage

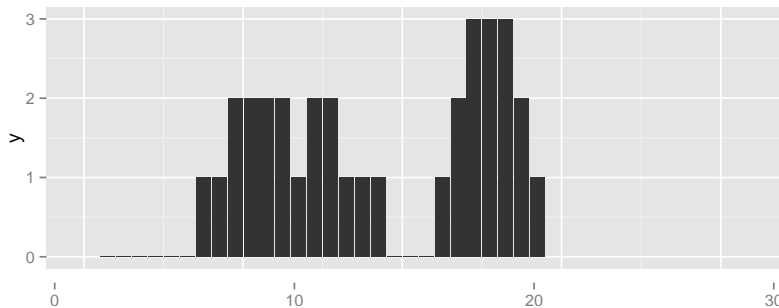
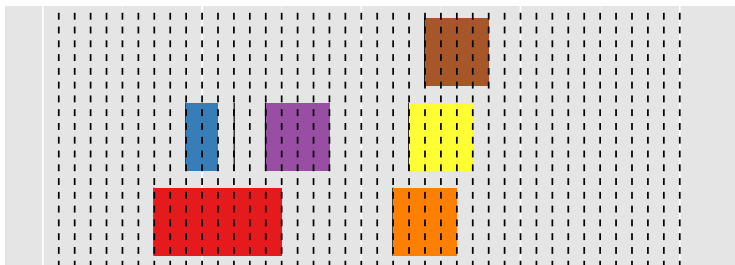
coverage returns a Run Length Encoding - an efficient representation of repeated values

```
cvrg <- coverage(ir)
cvrg

## integer-Rle of length 28 with 12 runs
##   Lengths: 6 2 4 1 2 3 3 1 1 3 1 1
##   Values  : 0 1 2 1 2 1 0 1 2 3 2 1

as.vector(cvrg[1:12])

##   [1] 0 0 0 0 0 0 0 1 1 2 2 2 2
```



'slice' to get peaks

```
ranges(slice(coverage(ir), 2))
```

```
## IRanges of length 3  
##      start end width  
## [1]      9  12     4  
## [2]     14  15     2  
## [3]     23  27     5
```

Overlaps...

e.g. counting

```
ir3 <- IRanges(start = c(1, 14, 27), end = c(13,  
18, 30))
```

```
ir3
```

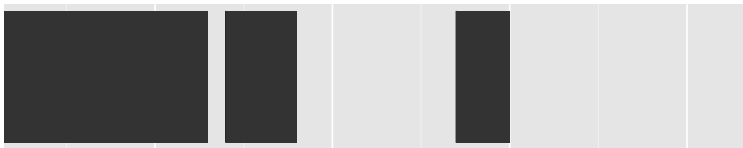
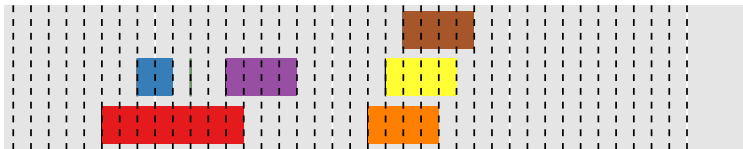
```
## IRanges of length 3
```

```
##      start end width
```

```
## [1]      1  13    13
```

```
## [2]     14  18     5
```

```
## [3]     27  30     4
```



Overlaps

```
query <- ir
subject <- ir3
ov <- findOverlaps(query, subject)
ov

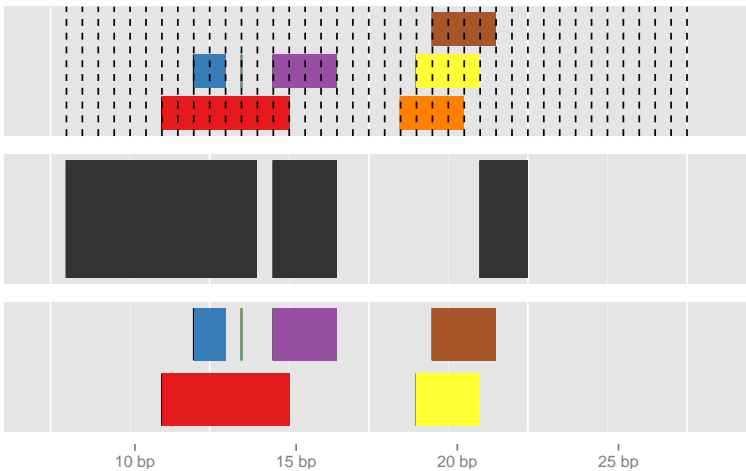
## Hits object with 7 hits and 0 metadata columns:
##      queryHits subjectHits
##      <integer>  <integer>
## [1]           1           1
## [2]           1           2
## [3]           2           1
## [4]           3           1
## [5]           4           2
## [6]           6           3
## [7]           7           3
## -----
## queryLength: 7
## subjectLength: 3
```

```
query[queryHits(ov)]
```

```
## IRanges of length 7
##      start end width
## [1]      7  15     9
## [2]      7  15     9
## [3]      9  11     3
## [4]     12  12     1
## [5]     14  18     5
## [6]     23  27     5
## [7]     24  28     5
```

```
subject[subjectHits(ov)]
```

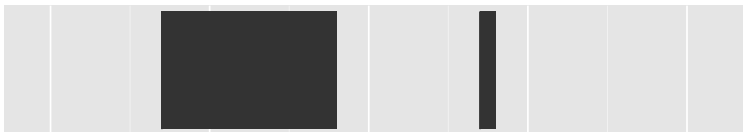
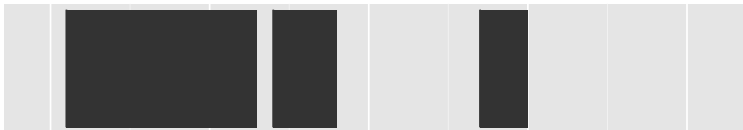
```
## IRanges of length 7
##      start end width
## [1]      1  13    13
## [2]     14  18     5
## [3]      1  13    13
## [4]      1  13    13
## [5]     14  18     5
## [6]     27  30     4
## [7]     27  30     4
```



Intersection

```
intersect(ir,ir3)

## IRanges of length 2
##      start end width
## [1]      7  18    12
## [2]     27  28     2
```



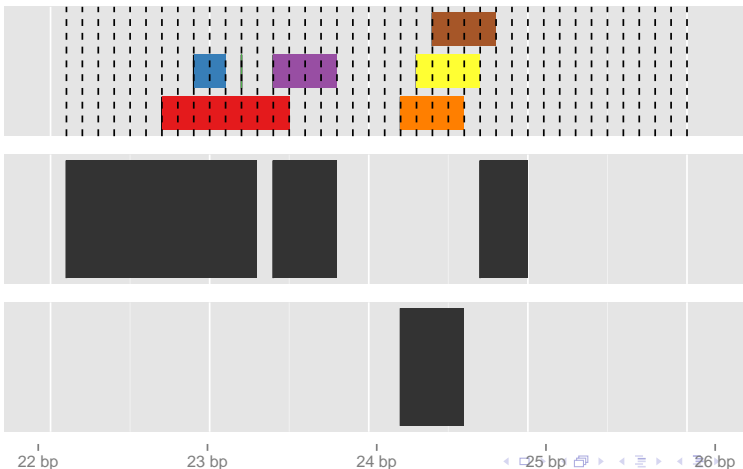
Subtraction

```
setdiff(ir,ir3)
```

```
## IRanges of length 1
```

```
##      start end width
```

```
## [1]    22  26     5
```



GRanges and Genomic Features

- ▶ GRanges provides infrastructure to manipulate genomic intervals in an efficient manner
- ▶ GenomicFeatures provides infrastructure to manipulate databases of genomic features (e.g. transcripts, exons)

We can define a 'chromosome' for each range

```
gr <- GRanges(c("A","A","A","B","B","B","B"), ranges=ir)
gr

## GRanges object with 7 ranges and 0 metadata columns:
##      seqnames      ranges strand
##      <Rle> <IRanges>  <Rle>
## [1]      A [ 7, 15]      *
## [2]      A [ 9, 11]      *
## [3]      A [12, 12]      *
## [4]      B [14, 18]      *
## [5]      B [22, 26]      *
## [6]      B [23, 27]      *
## [7]      B [24, 28]      *
## -----
## seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

Intermission

Work through section 4 of the practical

Reading alignments

We will assume that the sequencing reads have been aligned and that we are interested in processing the alignments. Rsamtools provides an interface for doing this. But we will use the readGAlignments tool in GenomicAlignments which extracts the essential information from the bam file.

```
bam <- readGAlignments(mybam,use.name=TRUE)
```

The result looks a lot like a GRanges object. In fact, a lot of the same operations can be use

```
bam[1:4]
```

```
## GAlignments object with 4 alignments and 0 metadata columns:
##           seqnames strand      cigar    qwidth
##           <Rle>  <Rle> <character> <integer>
## SRR031715.1138209   chr4      +       37M        37
## SRR031714.776678    chr4      -       37M        37
## SRR031715.3258011   chr4      -       37M        37
## SRR031715.4791418   chr4      +       37M        37
##           start      end      width    njunc
##           <integer> <integer> <integer> <integer>
## SRR031715.1138209    169      205      37        0
## SRR031714.776678    184      220      37        0
## SRR031715.3258011    187      223      37        0
## SRR031715.4791418    193      229      37        0
## -----
## seqinfo: 8 sequences from an unspecified genome
```

Querying alignments

```
table(strand(bam))
```

```
##
```

```
##      +      -      *
```

```
## 84871 90475      0
```

```
summary(width(bam))
```

```
##      Min.   1st Qu.   Median     Mean   3rd Qu.     Max.
```

```
##      37.00    37.00    37.00    58.72    37.00  19350.00
```

```
range(start(bam))
```

```
## [1]      169 1351760
```

```
cigar(bam)[1:10]
```

```
## [1] "37M" "37M" "37M" "37M" "37M" "37M" "37M" "37M" "37M" "37M"
```

```
## [10] "37M"
```

Manipulation of reads

The aligned reads can be manipulated using functions from IRanges

```
shift(ranges(bam),10)
```

```
## IRanges of length 175346
##           start      end width      names
## [1]         179      215    37 SRR031715.1138209
## [2]         194      230    37  SRR031714.776678
## [3]         197      233    37 SRR031715.3258011
## [4]         203      239    37 SRR031715.4791418
## [5]         336      372    37 SRR031715.1138209
## ...           ...      ...    ...           ...
## [175342] 1349718 1349754    37 SRR031714.1650928
## [175343] 1349848 1349884    37 SRR031714.1650928
## [175344] 1351650 1351686    37 SRR031714.5192891
## [175345] 1351650 1351686    37 SRR031715.2351056
## [175346] 1351770 1351806    37  SRR031714.864195
```

Manipulation of reads

The aligned reads can be manipulated using functions from IRanges

```
flank(ranges(bam), 100, both=T)
```

```
## IRanges of length 175346
```

```
##           start      end width      names
## [1]           69      268   200 SRR031715.1138209
## [2]           84      283   200  SRR031714.776678
## [3]           87      286   200 SRR031715.3258011
## [4]           93      292   200 SRR031715.4791418
## [5]          226      425   200 SRR031715.1138209
## ...           ...      ...   ...           ...
## [175342] 1349608 1349807   200 SRR031714.1650928
## [175343] 1349738 1349937   200 SRR031714.1650928
## [175344] 1351540 1351739   200 SRR031714.5192891
## [175345] 1351540 1351739   200 SRR031715.2351056
## [175346] 1351660 1351859   200  SRR031714.864195
```

```
coverage(ranges(bam))
```

```
## integer-Rle of length 1351796 with 104286 runs
```

```
##   Lengths: 168   15    3    6   13 ... 1765   37   83   37
```

Region subset - the naive way

```
bam[start(bam) < 20100 & end(bam) > 20000, ]
```

```
## GAlignments object with 14 alignments and 0 metadata columns:
```

```
##           seqnames strand      cigar    qwidth
##           <Rle>   <Rle> <character> <integer>
## SRR031714.4100693   chr4      +   31M7704N6M      37
## SRR031715.5248298   chr4      +   29M7704N8M      37
## SRR031714.4092638   chr4      -      37M          37
## SRR031714.4275537   chr4      -      37M          37
## SRR031715.1315719   chr4      -      37M          37
## ...               ...      ...      ...          ...
## SRR031715.3358559   chr4      +      37M          37
## SRR031715.4831822   chr4      +      37M          37
## SRR031715.4459351   chr4      +      37M          37
## SRR031715.2716654   chr4      -      37M          37
## SRR031715.1552693   chr4      +      37M          37
##           start      end      width    njunc
##           <integer> <integer> <integer> <integer>
## SRR031714.4100693   13660    21400    7741      1
## SRR031715.5248298   13662    21402    7741      1
## SRR031714.4092638   19968    20004     37        0
## SRR031714.4275537   19968    20004     37        0
```

The smart way

```
gr <- GRanges("chr4", IRanges(start = 20000, end = 20100))
gr

## GRanges object with 1 range and 0 metadata columns:
##      seqnames      ranges strand
##      <Rle>        <IRanges> <Rle>
## [1]      chr4 [20000, 20100]      *
## -----
##      seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

```
findOverlaps(gr,bam)
```

```
## Hits object with 12 hits and 0 metadata columns:
```

```
##           queryHits subjectHits
```

```
##           <integer>   <integer>
```

```
##      [1]           1         6699
```

```
##      [2]           1         6700
```

```
##      [3]           1         6701
```

```
##      [4]           1         6702
```

```
##      [5]           1         6703
```

```
##      ...           ...           ...
```

```
##      [8]           1         6706
```

```
##      [9]           1         6707
```

```
##     [10]           1         6708
```

```
##     [11]           1         6709
```

```
##     [12]           1         6710
```

```
## -----
```

```
## queryLength: 1
```

```
## subjectLength: 175346
```



```
bam[subjectHits(findOverlaps(gr,bam))]
```

```
## GAlignments object with 12 alignments and 0 metadata columns:
```

	seqnames	strand	cigar	qwidth	
	<Rle>	<Rle>	<character>	<integer>	
##	SRR031714.4092638	chr4	-	37M	37
##	SRR031714.4275537	chr4	-	37M	37
##	SRR031715.1315719	chr4	-	37M	37
##	SRR031715.1502533	chr4	-	37M	37
##	SRR031714.336402	chr4	-	37M	37
##
##	SRR031715.3358559	chr4	+	37M	37
##	SRR031715.4831822	chr4	+	37M	37
##	SRR031715.4459351	chr4	+	37M	37
##	SRR031715.2716654	chr4	-	37M	37
##	SRR031715.1552693	chr4	+	37M	37
##		start	end	width	njunc
##		<integer>	<integer>	<integer>	<integer>
##	SRR031714.4092638	19968	20004	37	0
##	SRR031714.4275537	19968	20004	37	0
##	SRR031715.1315719	19968	20004	37	0
##	SRR031715.1502533	19968	20004	37	0
##	SRR031714.336402	19971	20007	37	0
##

Alternative

```
bam.sub <- bam[bam %over% gr]
```

```
bam.sub
```

```
## GAlignments object with 12 alignments and 0 metadata columns:
```

```
##           seqnames strand      cigar  qwidth
##           <Rle>  <Rle> <character> <integer>
## SRR031714.4092638   chr4      -      37M      37
## SRR031714.4275537   chr4      -      37M      37
## SRR031715.1315719   chr4      -      37M      37
## SRR031715.1502533   chr4      -      37M      37
## SRR031714.336402    chr4      -      37M      37
## ...                ...      ...      ...      ...
## SRR031715.3358559   chr4      +      37M      37
## SRR031715.4831822   chr4      +      37M      37
## SRR031715.4459351   chr4      +      37M      37
## SRR031715.2716654   chr4      -      37M      37
## SRR031715.1552693   chr4      +      37M      37
##           start      end      width  njunc
##           <integer> <integer> <integer> <integer>
## SRR031714.4092638   19968    20004     37      0
## SRR031714.4275537   19968    20004     37      0
## SRR031715.1315719   19968    20004     37      0
```

Read subset of regions

Quicker still, we can get the reads directly from the bam file. The region to be read can be specified using the param argument.

```
system.time(bam.sub <-  
readGAlignments(file=mybam,  
use.names=TRUE,  
param=ScanBamParam(which=gr)))
```

```
##      user  system elapsed  
##    0.057    0.001    0.056
```

Recap

- ▶ Ranges can be used to represent continuous regions
- ▶ GRanges are special ranges with extra biological context
- ▶ GRanges can be manipulated, compared, overlapped with each other
- ▶ Aligned reads can be represented by Ranges
- ▶ Genome and sequencing reads can be represented efficiently by Biostrings
- ▶ The genome can also be accessed using Ranges

Now, work through Section 5 of the practical