Bloconductor |Bioinformatics Basics

Code

Hide

```
# This R environment comes with many helpful analytics packages installed
# It is defined by the kaggle/rstats Docker image: https://github.com/kaggle/docker-rstats
# For example, here's a helpful package to load

library(tidyverse) # metapackage of all tidyverse packages

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under
 the input directory

list.files(path = "C:/Users/samen/Desktop/Bioinformatics Projects/Bioconductor tools for Mass Sp
ectrometry/Bioconductor")
```

```
 [1] "alignments"                    "bioconductor-introduction.ipynb"
 [3] "Bioconductor.Rproj"            "blast_queries"
 [5] "machine_learning"              "output"
 [7] "renv"                          "renv.lock"
 [9] "SCE"                           "sequences"
[11] "substitution_matrices"
```

Hide

```
# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as ou
tput when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the cu
rrent session
```

Hide

```
suppressWarnings(expr)
```

```
function (expr)
{
    enexpr(expr)
}
<bytecode: 0x0000024846035978>
<environment: namespace:rlang>
```

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.

Hide

```
#packages installation
if (!requireNamespace("BiocManager", quietly= TRUE))
    install.packages("BioManager")
BiocManager:: install("Biostrings")
```

```
'getOption("repos")' replaces Bioconductor standard repositories, see
'?repositories' for details

replacement repositories:
    CRAN: https://cran.rstudio.com

Bioconductor version 3.14 (BiocManager 1.30.17), R 4.1.2 (2021-11-01)
Warning: package(s) not installed when version(s) same as current; use `force =
  TRUE` to re-install: 'Biostrings'
Installation paths not writeable, unable to update packages
  path: C:/Program Files/R/R-4.1.2/library
  packages:
    class, cluster, foreign, MASS, Matrix, mgcv, nlme, nnet, rpart,
    spatial, survival
Old packages: 'cli', 'dplyr', 'MSnbase', 'RSQLite'
```

```
Update all/some/none? [a/s/n]:
```

Hide

```
n
BiocManager:: install("msa")
```

```
'getOption("repos")' replaces Bioconductor standard repositories, see
'?repositories' for details

replacement repositories:
    CRAN: https://cran.rstudio.com

Bioconductor version 3.14 (BiocManager 1.30.17), R 4.1.2 (2021-11-01)
Warning: package(s) not installed when version(s) same as current; use `force =
  TRUE` to re-install: 'msa'
Installation paths not writeable, unable to update packages
  path: C:/Program Files/R/R-4.1.2/library
  packages:
    class, cluster, foreign, MASS, Matrix, mgcv, nlme, nnet, rpart,
    spatial, survival
Old packages: 'cli', 'dplyr', 'MSnbase', 'RSQLite'
```

```
Update all/some/none? [a/s/n]:
```

Hide

```
n
```

BIOCONDUCTOR¶ Bioconductor is quite more advanced compared to say Biopython & requires minimal programming on the user end. I have covered some basic sequence operations in a biopython notebook or Working with Sequences noteobook on a relatable topic. The libraries used in this notebook:

I. Biostrings (General base library for work with strings, uses FASTA for imports) (II) msa (Library for multiple sequence alignment, containing more advanced methods than the progressive approach covered in biological sequence alignment)

Hide

```
#Bioconductor:: Biostrings
#import library without messages

suppressPackageStartupMessages(library(Biostrings))
```

Hide

```
#Sequence Operations

#1 Defining characters of DNA and amino acids
chr_n1 = "ACTTCACCAGCTCCCTGGCGGTAAGTTGATCAAAGGAAAC"
chr_n2 = "TTTCGGGTAAGTAAATATATGTTTCACTACTTCCTTTCGG"

chr_aa1 = 'PAWHEAE'
chr_aa2 = 'HEAGAWGHEE'


# Nucleotide String
s1_n <- DNAString(chr_n1) #DNAString
s2_n <- DNAString(chr_n2)
s2_n
```

```
40-letter DNAString object
seq: TTTCGGGTAAGTAAATATATGTTTCACTACTTCCTTTCGG
```

Hide

```
#Amino Acid String
s1_aa = AAString(chr_aa1)
s2_aa = AAString(chr_aa2)
s2_aa
```

```
10-letter AAString object
seq: HEAGAWGHEE
```

Hide

```
#Define a new XstringSet from characters (3 sequences)

#concat to make vector with c()
str_concat = c("ACGT","GTCA","GCTA")
n_set <- AAStringSet(str_concat)
n_set
```

```
AAStringSet object of length 3:
    width seq
[1]     4 ACGT
[2]     4 GTCA
[3]     4 GCTA
```

Hide

```
#Define a new XStringSet from characters (1 sequence)

n_set_1 <- DNAStringSet(c("ACGT"))
n_set_1
```

```
DNAStringSet object of length 1:
    width seq
[1]     4 ACGT
```

Hide

```
#Create a stringset from a sequence string
#Using DNAString -> DNAStringSet

str_strset = DNAStringSet(s1_n)
```

Hide

```
# Start with set (just the one)
string = n_set[1]
string
```

```
AAStringSet object of length 1:
    width seq
[1]     4 ACGT
```

Hide

```
#Convert XStringSet to Character
dna_char <- toString(n_set[1])
class(dna_char) #check the class type
```

```
[1] "character"
```

Hide

```
dna_char #print character
```

```
[1] "ACGT"
```

Hide

```
#start with many strings in a stringset
print(n_set)
```

```
AAStringSet object of length 3:
    width seq
[1]     4 ACGT
[2]     4 GTCA
[3]     4 GCTA
```

Hide

```
lst <- list() #defines an empty list

#loop through allin n_set
for(i in 1:length(n_set)) {
    lst <- c(lst, toString(n_set[i]))
}

lst  #list containing characters
```

```
[[1]]
[1] "ACGT"

[[2]]
[1] "GTCA"

[[3]]
[1] "GCTA"
```

Hide

```
# Set - > Single sequence
string = n_set[[1]] # extract single sequence
string # print string
```

```
4-letter AAString object
seq: ACGT
```

Hide

```
# use toString
char = toString(string)
char  # print character
```

```
[1] "ACGT"
```

Hide

```
class(char) # print char type
```

```
[1] "character"
```

’‘READING SEQUENCES FROM FASTA FILE¶ Usually when working with realistic sequences formats such as FASTA & GenBank are used Biostrings uses the FASTA format for operations, loading & saving. The two class formats used upon the sequence(s) being read: DNAStringSet for nucleotide sequence set (even just the one) AAStringSet for amino acid sequences’”

Hide

```
# File Containing One Sequence
fasta_n = readDNAStringSet('C:/Users/samen/Desktop/Bioinformatics Projects/Bioconductor tools fo
r Mass Spectrometry/Bioconductor/sequences/example.fasta')
fasta_n # print read data
```

```
DNAStringSet object of length 1:
    width seq                                          names
[1]  1231 GGCAGATTCCCCCTAGACC...CCCAAATAAACTCCAGAAG HSBGPG Human gene...
```

Hide

```
class(fasta_n) # print read class format
```

```
[1] "DNAStringSet"
attr(,"package")
[1] "Biostrings"
```

Hide

```
names(fasta_n) # print name of sequence
```

```
[1] "HSBGPG Human gene for bone gla protein (BGP)"
```

Hide

```
# can use (Biostrings::) prefix as well
fasta_aa = Biostrings::readAAStringSet('C:/Users/samen/Desktop/Bioinformatics Projects/Bioconduc
tor tools for Mass Spectrometry/Bioconductor/sequences/NC_005816.faa')
fasta_aa
```

```
AAStringSet object of length 10:
     width seq                                          names
 [1]   340 MVTFETVMEIKILHKQGMS...HPLHHPLSIYDSFCRGVA gi|45478712|ref|N...
 [2]   260 MMMELQHQRLMALAGQLQL...YRLRQKRKAGVIAEANPE gi|45478713|ref|N...
 [3]    64 MNKQQQTALNMARFIRSQS...ELQNSIQARFEAESETGT gi|45478714|ref|N...
 [4]   123 MSKKRRPQKRPRRRRFFHR...FSPTTAPYPVTIVLSPTR gi|45478715|ref|N...
 [5]   145 MGGGMISKLFCLALIFLSS...IVVKEIKKSIPGCTVYYH gi|45478716|ref|N...
 [6]   357 MSDTMVVNGSGGVPAFLFS...RKREGALVQKDIDSGLLK gi|45478717|ref|N...
 [7]   138 MKFHFCDLNHSYKNQEGKI...KKPEGVEPREGQEREDLP gi|45478718|ref|N...
 [8]   312 MKKSSIVATIITILSGSAN...AGISNKNYTVTAGLQYRF gi|45478719|ref|N...
 [9]    99 MRTLDEVIASRSPESQTRI...KLSLDVELPTGRRVAFHV gi|45478720|ref|N...
[10]    90 MADLKKLQVYGPELPRPYA...VRIAEDEFTAHLNTLESK gi|45478721|ref|N...
```

Hide

```
class(fasta_aa) # AAStringSet object
```

```
[1] "AAStringSet"
attr(,"package")
[1] "Biostrings"
```

Hide

```
#always start with 1, not a 0 like python
fasta_aa[1] #Still AA stringset object but length of 1
```

```
AAStringSet object of length 1:
    width seq                                          names
[1]   340 MVTFETVMEIKILHKQGMS...KHPLHHPLSIYDSFCRGVA gi|45478712|ref|N...
```

Hide

```
#Other operations of fast.aa files
width(fasta_aa[1]) #get length of sequence
```

```
[1] 340
```

Hide

```
seq(fasta_aa[1]) #sequence number
```

```
[1] 1
```

Hide

```
names (fasta_aa[1]) #get the character object type of the sequence
```

```
[1] "gi|45478712|ref|NP_995567.1| putative transposase [Yersinia pestis biovar Microtus str. 910
01]"
```

Hide

```
class(char) #show object class
```

```
[1] "character"
```

'"SAVING SEQUENCES TO FASTA FORMAT writeXStringSet is used to save a StringSet, which has the option to save in FASTA format"'

Hide

```
n_set #an aastringset we wish to save
```

```
AAStringSet object of length 3:
    width seq
[1]     4 ACGT
[2]     4 GTCA
[3]     4 GCTA
```

Hide

```
#Save XStringSet
writeXStringSet(n_set, filepath = 'C:/Users/samen/Desktop/Bioinformatics Projects/Bioconductor t
ools for Mass Spectrometry/Bioconductor/output/dna_list.fasta', format = 'fasta' )
```

Hide

```
#confirmation only (read the file)
confirm_dna_xstrset = readDNAStringSet ('C:/Users/samen/Desktop/Bioinformatics Projects/Biocondu
ctor tools for Mass Spectrometry/Bioconductor/output/dna_list.fasta')

confirm_dna_xstrset
```

```
DNAStringSet object of length 3:
    width seq                                          names
[1]     4 ACGT
[2]     4 GTCA
[3]     4 GCTA
```

Hide

```
# combine characters
x0 <- DNAStringSet(c("CTCCCAGTAT", "TTCCCGA", "TACCTAGAG"))  # String Set #1
x1 <- DNAStringSet(c("AGGTCGT", "GTCAGTGGTCCCC", "CATTTTAGG")) # String Set #2
x2 <- DNAStringSet(c("TGCTAGCTA", "AGTCTTGC", "AGCTTTCGAG")) # String Set #3

dna_list <- list(x0, x1, x2) # create a list of String Sets
dna_xstrset = do.call(c, dna_list) # concentrate
dna_xstrset
```

```
DNAStringSet object of length 9:
    width seq
[1]    10 CTCCCAGTAT
[2]     7 TTCCCGA
[3]     9 TACCTAGAG
[4]     7 AGGTCGT
[5]    13 GTCAGTGGTCCCC
[6]     9 CATTTTAGG
[7]     9 TGCTAGCTA
[8]     8 AGTCTTGC
[9]    10 AGCTTTCGAG
```

Hide

```
#Select only specific sequences from Set
dna_xstrset[1:2] #indexing a Set -> selecting sequences
```

```
DNAStringSet object of length 2:
    width seq
[1]    10 CTCCCAGTAT
[2]     7 TTCCCGA
```

Hide

```
new_set <- dna_xstrset[9] #set to new variable
```

Hide

```
# Selecting Sequence Subset w/ range
subseq_aa = subseq(s2_aa, start=1,end=5)
subseq_aa
```

```
5-letter AAString object
seq: HEAGA
```

”’1.4 | BASIC FUNCTIONALITY

Some basic functions appliable to StringSet, some of which have not been used yet, mainly to do with ordering or visualisation inside the set ”’

```
<!-- rnb-text-end -->


<!-- rnb-chunk-begin -->


<!-- rnb-source-begin eyJkYXRhIjoiYGBgclxuI29wZXJhdGlvbnMgdXNpbmcgRE5BU3RyaW5nIGFuZCBBQVN0cmluZy
BPYmplY3RzXG5zMV9yZXZlcnNlIDwtIHJldmVyc2UoczFfbilcbnMxX2NvbXBsZW1lbnQgPC0gY29tcGxlbWVudChzMV9uKV
xuczFfcmV2ZXJzZWNvbXBsZW1lbnQgPSByZXZlcnNlQ29tcGxlbWVudChzMV9uKVxuXG5jKHMxX3JldmVyc2UpXG5gYGAifQ
== -->

```r
#operations using DNAString and AAString Objects
s1_reverse <- reverse(s1_n)
s1_complement <- complement(s1_n)
s1_reversecomplement = reverseComplement(s1_n)

c(s1_reverse)
```

```
40-letter DNAString object
seq: CAAAGGAAACTAGTTGAATGGCGGTCCCTCGACCACTTCA
```

Hide

```
c(s1_complement)
```

```
40-letter DNAString object
seq: TGAAGTGGTCGAGGGACCGCCATTCAACTAGTTTCCTTTG
```

Hide

```
c(s1_reversecomplement)
```

```
40-letter DNAString object
seq: GTTTCCTTTGATCAACTTACCGCCAGGGAGCTGGTGAAGT
```

Hide

```
#Same goes for DNAStringSet class sequences
class(fasta_n) #check class
```

```
[1] "DNAStringSet"
attr(,"package")
[1] "Biostrings"
```
```

Hide

```
s1_reverse_xstr = reverse(fasta_n)
s1_reverse_xstr
```

```
DNAStringSet object of length 1:
    width seq                                          names
[1]  1231 GAAGACCTCAAATAAACCC...CCAGATCCCCCTTAGACGG HSBGPG Human gene...
```

Hide

```
# Translation works w/ Sets or just the XString
s1_translate <- translate(dna_xstrset[[3]], no.init.codon=TRUE)
s1_translate
```

```
3-letter AAString object
seq: YLE
```

Hide

```
alphabetFrequency(DNAString(s1_complement))
```

```
 A  C  G  T  M  R  W  S  Y  K  V  H  D  B  N  -  +  .
 8  9 11 12  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Hide

```
#calculate the alphabet frequency of a DNA sequence
```

Hide

```
uniqueLetters(dna_xstrset[1])
```

```
[1] "A" "C" "G" "T"
```

Hide

```
#show all unique characters in a sequence
```

''' 1.5 | BIOLOGICAL FUNCTIONS

Biological functality relating to DNA is found in Biostrings as well Having one of the strands, we can get its reverse, complement & reverse complement, similar to that was shown in notebook Biological Sequence Operations Translation from DNA (or RNA) to chains of amino acids / proteins can be done via translate Translation works with both strings & string set objects '''

```
<!-- rnb-text-end -->


<!-- rnb-chunk-begin -->


<!-- rnb-source-begin eyJkYXRhIjoiYGBgclxuIyBDaGFyYWN0ZXIgZnJlcXVlbmN5IGZ1bmN0aW9uc1xuc2VxdWVuY2UgPC0gZG5hX3hzdHJzZXRbMV1cbnNlcXVlbmNlXG5gYGAifQ== -->

```r
# Character frequency functions
sequence <- dna_xstrset[1]
sequence
```

```
DNAStringSet object of length 1:
    width seq
[1]    10 CTCCCAGTAT
```

Hide

```
dinucleotideFrequency(sequence)
```

```
     AA AC AG AT CA CC CG CT GA GC GG GT TA TC TG TT
[1,]  0  0  1  1  1  2  0  1  0  0  0  1  1  1  0  0
```

Hide

```
trinucleotideFrequency(sequence)
```

```
     AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA ATC ATG ATT
[1,]   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
     CAA CAC CAG CAT CCA CCC CCG CCT CGA CGC CGG CGT CTA CTC CTG CTT
[1,]   0   0   1   0   1   1   0   0   0   0   0   0   0   1   0   0
     GAA GAC GAG GAT GCA GCC GCG GCT GGA GGC GGG GGT GTA GTC GTG GTT
[1,]   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0
     TAA TAC TAG TAT TCA TCC TCG TCT TGA TGC TGG TGT TTA TTC TTG TTT
[1,]   0   0   0   1   0   1   0   0   0   0   0   0   0   0   0   0
```

Hide

```
oligonucleotideFrequency(sequence,width=2)
```

```
     AA AC AG AT CA CC CG CT GA GC GG GT TA TC TG TT
[1,]  0  0  1  1  1  2  0  1  0  0  0  1  1  1  0  0
```
```

Hide

```
oligonucleotideFrequency(sequence,width=4)
```

```
     AAAA AAAC AAAG AAAT AACA AACC AACG AACT AAGA AAGC AAGG AAGT AATA
[1,]    0    0    0    0    0    0    0    0    0    0    0    0    0
     AATC AATG AATT ACAA ACAC ACAG ACAT ACCA ACCC ACCG ACCT ACGA ACGC
[1,]    0    0    0    0    0    0    0    0    0    0    0    0    0
     ACGG ACGT ACTA ACTC ACTG ACTT AGAA AGAC AGAG AGAT AGCA AGCC AGCG
[1,]    0    0    0    0    0    0    0    0    0    0    0    0    0
     AGCT AGGA AGGC AGGG AGGT AGTA AGTC AGTG AGTT ATAA ATAC ATAG ATAT
[1,]    0    0    0    0    0    1    0    0    0    0    0    0    0
     ATCA ATCC ATCG ATCT ATGA ATGC ATGG ATGT ATTA ATTC ATTG ATTT CAAA
[1,]    0    0    0    0    0    0    0    0    0    0    0    0    0
     CAAC CAAG CAAT CACA CACC CACG CACT CAGA CAGC CAGG CAGT CATA CATC
[1,]    0    0    0    0    0    0    0    0    0    0    1    0    0
     CATG CATT CCAA CCAC CCAG CCAT CCCA CCCC CCCG CCCT CCGA CCGC CCGG
[1,]    0    0    0    0    1    0    1    0    0    0    0    0    0
     CCGT CCTA CCTC CCTG CCTT CGAA CGAC CGAG CGAT CGCA CGCC CGCG CGCT
[1,]    0    0    0    0    0    0    0    0    0    0    0    0    0
     CGGA CGGC CGGG CGGT CGTA CGTC CGTG CGTT CTAA CTAC CTAG CTAT CTCA
[1,]    0    0    0    0    0    0    0    0    0    0    0    0    0
     CTCC CTCG CTCT CTGA CTGC CTGG CTGT CTTA CTTC CTTG CTTT GAAA GAAC
[1,]    1    0    0    0    0    0    0    0    0    0    0    0    0
     GAAG GAAT GACA GACC GACG GACT GAGA GAGC GAGG GAGT GATA GATC GATG
[1,]    0    0    0    0    0    0    0    0    0    0    0    0    0
     GATT GCAA GCAC GCAG GCAT GCCA GCCC GCCG GCCT GCGA GCGC GCGG GCGT
[1,]    0    0    0    0    0    0    0    0    0    0    0    0    0
     GCTA GCTC GCTG GCTT GGAA GGAC GGAG GGAT GGCA GGCC GGCG GGCT GGGA
[1,]    0    0    0    0    0    0    0    0    0    0    0    0    0
     GGGC GGGG GGGT GGTA GGTC GGTG GGTT GTAA GTAC GTAG GTAT GTCA GTCC
[1,]    0    0    0    0    0    0    0    0    0    0    1    0    0
     GTCG GTCT GTGA GTGC GTGG GTGT GTTA GTTC GTTG GTTT TAAA TAAC TAAG
[1,]    0    0    0    0    0    0    0    0    0    0    0    0    0
     TAAT TACA TACC TACG TACT TAGA TAGC TAGG TAGT TATA TATC TATG TATT
[1,]    0    0    0    0    0    0    0    0    0    0    0    0    0
     TCAA TCAC TCAG TCAT TCCA TCCC TCCG TCCT TCGA TCGC TCGG TCGT TCTA
[1,]    0    0    0    0    0    1    0    0    0    0    0    0    0
     TCTC TCTG TCTT TGAA TGAC TGAG TGAT TGCA TGCC TGCG TGCT TGGA TGGC
[1,]    0    0    0    0    0    0    0    0    0    0    0    0    0
     TGGG TGGT TGTA TGTC TGTG TGTT TTAA TTAC TTAG TTAT TTCA TTCC TTCG
[1,]    0    0    0    0    0    0    0    0    0    0    0    0    0
     TTCT TTGA TTGC TTGG TTGT TTTA TTTC TTTG TTTT
[1,]    0    0    0    0    0    0    0    0    0
```

Hide

```
#Similar to Pandas, if the list is too long,the default view will ...
#'options' can be used to change the maximum column count

options(repr.matrix.max.cols = 70,
        repr.matrix.max.rows = 100)
```

'''1.6 | COUNTING CHARACTERS

Sequence alphabet counts are quite relevant in bioinformatics, eg. GC Content is the dinucleotide count Other sequence alphabet counters:

alphabetFrequency - For a general alphabet count of the sequence/set dinucleotideFrequency - For two character pair counts trinucleotideFrequency - For three character pair counts (codons) oligonucleotideFrequency - General form of the three above & beyond, description below: Oligonucleotides | ScienceDirect

Oligonucleotides are small molecules 8–50 nucleotides in length that bind via Watson-Crick base pairing to enhance or repress the expression of target RNA ''''

Hide

```
trinucleotideFrequency(dna_xstrset[1])
```

```
     AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA ATC ATG ATT
[1,]   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
     CAA CAC CAG CAT CCA CCC CCG CCT CGA CGC CGG CGT CTA CTC CTG CTT
[1,]   0   0   1   0   1   1   0   0   0   0   0   0   0   1   0   0
     GAA GAC GAG GAT GCA GCC GCG GCT GGA GGC GGG GGT GTA GTC GTG GTT
[1,]   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0
     TAA TAC TAG TAT TCA TCC TCG TCT TGA TGC TGG TGT TTA TTC TTG TTT
[1,]   0   0   0   1   0   1   0   0   0   0   0   0   0   0   0   0
```

Hide

```
#calculating consensus matrix for a string set
dna_xstrset
```

```
DNAStringSet object of length 9:
    width seq
[1]    10 CTCCCAGTAT
[2]     7 TTCCCGA
[3]     9 TACCTAGAG
[4]     7 AGGTCGT
[5]    13 GTCAGTGGTCCCC
[6]     9 CATTTTAGG
[7]     9 TGCTAGCTA
[8]     8 AGTCTTGC
[9]    10 AGCTTTCGAG
```

Hide

```
consensusMatrix(dna_xstrset, as.prob = FALSE)
```

```
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
A    3    2    0    1    1    2    2    1    3     0     0     0     0
C    2    0    6    4    3    0    2    1    0     1     1     1     1
G    1    4    1    0    1    3    4    3    2     1     0     0     0
T    3    3    2    4    4    4    1    2    1     1     0     0     0
M    0    0    0    0    0    0    0    0    0     0     0     0     0
R    0    0    0    0    0    0    0    0    0     0     0     0     0
W    0    0    0    0    0    0    0    0    0     0     0     0     0
S    0    0    0    0    0    0    0    0    0     0     0     0     0
Y    0    0    0    0    0    0    0    0    0     0     0     0     0
K    0    0    0    0    0    0    0    0    0     0     0     0     0
V    0    0    0    0    0    0    0    0    0     0     0     0     0
H    0    0    0    0    0    0    0    0    0     0     0     0     0
D    0    0    0    0    0    0    0    0    0     0     0     0     0
B    0    0    0    0    0    0    0    0    0     0     0     0     0
N    0    0    0    0    0    0    0    0    0     0     0     0     0
-    0    0    0    0    0    0    0    0    0     0     0     0     0
+    0    0    0    0    0    0    0    0    0     0     0     0     0
.    0    0    0    0    0    0    0    0    0     0     0     0     0
```

Hide

```
#Two sequences to be globally aligned

s1_n
```

```
40-letter DNAString object
seq: ACTTCACCAGCTCCCTGGCGGTAAGTTGATCAAAGGAAAC
```

Hide

```
s2_n
```

```
40-letter DNAString object
seq: TTTCGGGTAAGTAAATATATGTTTCACTACTTCCTTTCGG
```

Hide

```
# Nucleotide Global Alignment

#Define our own substition matrix (nucleotide)

mat <- nucleotideSubstitutionMatrix(match = 1, mismatch = -3,
                                    baseOnly = TRUE)

mat
```

```
   A  C  G  T
A  1 -3 -3 -3
C -3  1 -3 -3
G -3 -3  1 -3
T -3 -3 -3  1
```

Hide

```
class(mat)
```

```
[1] "matrix" "array"
```

Hide

```
#Global Alignment (Needleman Wunsch)
globalAlign <- pairwiseAlignment(s1_n, s2_n, #sequences we want to align
                                 type = 'global', #type of alignment
                                 substitutionMatrix = mat, #substitution matrix
                                gapOpening = 5, gapExtension =2
                                #gap penalty arguments
                                )

globalAlign
```

```
Global PairwiseAlignmentsSingleSubject (1 of 1)
pattern: ACTTCACCAGCTCCCTGGCGGTAAGTTGATCAAAGGAAAC------
subject: TTT----CGGGTAAGTAAATATATGTT--TCACTACTTCCTTTCGG
score: -85
```

Hide

```
#NUCLEOTIDE LOCAL SEQUENCE ALIGNMENT
#Smith-Waterman local sequence alignment between two nucleotide sequences s1_n & s2_n

#Nucleotide Local Sequence Alignment (Smith-Waterman)

localAlign <- pairwiseAlignment(s1_n, s2_n, type = "local",
                                substitutionMatrix = mat,
                                gapOpening= 5, gapExtension = 2)

localAlign
```

```
Local PairwiseAlignmentsSingleSubject (1 of 1)
pattern: [20] GGTAAGT
subject:  [6] GGTAAGT
score: 7
```

Hide

```
#Protein Global Alignment
#Needleman-Wunsch global sequence alignment between two amino acid chain sequences
#s1_aa and s2_aa
#global alignment(default type) using BLOSUM Substitution mAtrix

#45, 50,62, 80,100

pairwiseAlignment(s1_aa, s2_aa, substitutionMatrix = "BLOSUM62",
                  gapOpening = 0, gapExtension = 8)
```

```
Global PairwiseAlignmentsSingleSubject (1 of 1)
pattern: -PA--WHEAE
subject: HEAGAWGHEE
score: -8
```

”’2 | PAIRWISE SEQUENCE ALIGNMENT¶ Given the significance of PSA in various application of bioinformatics, we will look at quite a few things that are associated with this part of the library.

The gap penalties are regulated by the gapOpening and gapExtension arguments First we need to define aspects of our objective function; substitution matrix & gap penalties Gap penalties are specified in pairwiseAlignment, whilst the substitution matrix is created or called separately nucleotideSubstitutionMatrix - Create a substitution matrix w/ a match & mismatches in a nucleotide sequence or use strings to call preset aa matrices pairwiseAlignment - sequence alignment, by default global option is set Similar to python, long strings will contain …: To display the whole sequence we can use alignedPattern & alignedSubject together with c() ”’

```
'''2.1 | ALIGNMENT EXAMPLES

NUCLEOTIDE GLOBAL SEQUENCE ALIGNMENT
Nucleotide global sequence alignment using the Needleman Wunsch algorithm
We can set a self defined substitution matrix (constant match/mismatch) using nucleotideSubstitu
tionMatrix
pairwiseAlignment requires arguments type= ''global'', substitutionMatrix (mat) & gap model sett
ings (gapOpening,gapExtension) '''
```

Hide

```
#global alignment (default type) using PAM substituion Matrix

#30,40,70,120,250
pairwiseAlignment(s1_aa, s2_aa,
                  substitutionMatrix = 'PAM250',
                  gapOpening = 0, gapExtension = 1)
```

```
Global PairwiseAlignmentsSingleSubject (1 of 1)
pattern: --P-AW-HEAE
subject: HEAGAWGHE-E
score: 29
```

Hide

```
#Extracting Data from Alignments
#getting individual sequence in the alignment, alignedPattern and alignedSubject in StringSet ob
ject format

#sequence extraction
s1_nset = DNAStringSet(chr_n1)
s2_nset = DNAStringSet(chr_n2)

#Pairwise Sequence Alignment operation
alg <- pairwiseAlignment(s1_nset, s2_nset)

#recalling the sequences in a pairwise alignment
alignedPattern(alg)
```

```
DNAStringSet object of length 1:
    width seq
[1]    46 ACTTCACCAGCTCCCTGGCGGTAAGTTGATCAAAGGAAAC------
```

Hide

```
toString(alignedSubject(alg)) #convert string
```

```
[1] "TTT----CGGGTAAGTAAATATATGTT--TCACTACTTCCTTTCGG"
```

Hide

```
#summary of alignment
summary(alg)
```

```
Global Single Subject Pairwise Alignments
Number of Alignments:  1

Scores:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -168.2  -168.2  -168.2  -168.2  -168.2  -168.2

Number of matches:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
     14      14      14      14      14      14

Top 10 Mismatch Counts:
```

SubjectPosition <int>	Subject <chr>	Pattern <chr>	Count <int>	Probability <dbl>
1	T	A	1	1
2	T	C	1	1

SubjectPosition <int>	Subject <chr>	Pattern <chr>	Count <int>	Probability <dbl>
5	G	A	1	1
7	G	C	1	1
9	A	C	1	1
10	A	C	1	1
11	G	C	1	1
13	A	G	1	1
14	A	G	1	1
15	A	C	1	1

1-10 of 10 rows

Hide

```
globalAlign
```

```
Global PairwiseAlignmentsSingleSubject (1 of 1)
pattern: ACTTCACCAGCTCCCTGGCGGTAAGTTGATCAAAGGAAAC------
subject: TTT----CGGGTAAGTAAATATATGTT--TCACTACTTCCTTTCGG
score: -85
```

Hide

```
# Other alignment related functions

alphabet(globalAlign) # show characters of alignment sequences
```

```
 [1] "A" "C" "G" "T" "M" "R" "W" "S" "Y" "K" "V" "H" "D" "B" "N" "-" "+"
[18] "."
```

Hide

```
compareStrings(globalAlign) # compare strings of sequences
```

```
[1] "??T++++C?G?T???T?????TA?GTT++TCA???????C"
```

Hide

```
deletion(globalAlign)
```

```
IRangesList object of length 1:
[[1]]
IRanges object with 0 ranges and 0 metadata columns:
       start       end     width
   <integer> <integer> <integer>
```

Hide

```
mismatchTable(globalAlign)
```

PatternId <int>	PatternStart <int>	PatternEnd <int>	PatternSubstring <chr>	SubjectStart <int>	SubjectEnd <int>	SubjectSub <chr>
1	1	1	A	1	1	T
1	2	2	C	2	2	T
1	9	9	A	5	5	G
1	11	11	C	7	7	G
1	13	13	C	9	9	A
1	14	14	C	10	10	A
1	15	15	C	11	11	G
1	17	17	G	13	13	A
1	18	18	G	14	14	A
1	19	19	C	15	15	A

1-10 of 20 rows Previous **1** 2 Next

Hide

```
nchar(globalAlign)
```

```
[1] 40
```

Hide

```
nedit(globalAlign)
```

```
[1] 26
```

Hide

```
indel(globalAlign)
```

```
An object of class "InDel"
Slot "insertion":
IRangesList object of length 1:
[[1]]
IRanges object with 2 ranges and 0 metadata columns:
          start       end     width
      <integer> <integer> <integer>
  [1]         4         7         4
  [2]        24        25         2


Slot "deletion":
IRangesList object of length 1:
[[1]]
IRanges object with 0 ranges and 0 metadata columns:
       start       end     width
   <integer> <integer> <integer>
```

Hide

```
insertion(globalAlign)
```

```
IRangesList object of length 1:
[[1]]
IRanges object with 2 ranges and 0 metadata columns:
          start       end     width
      <integer> <integer> <integer>
  [1]         4         7         4
  [2]        24        25         2
```

Hide

```
nindel(globalAlign)
```

```
An object of class "InDel"
Slot "insertion":
     Length WidthSum
[1,]      2        6

Slot "deletion":
     Length WidthSum
[1,]      0        0
```

Hide

```
nmatch(globalAlign)
```

```
[1] 14
```

Hide

```
nmismatch(globalAlign)
```

```
[1] 20
```

Hide

```
pattern(globalAlign) # show only pattern sequence
```

```
[1] ACTTCACCAGCTCCCTGGCGGTAAGTTGATCAAAGGAAAC
```

Hide

```
subject(globalAlign) # show only subject sequence
```

```
[1] TTT----CGGGTAAGTAAATATATGTT--TCACTACTTCC
```

Hide

```
pid(globalAlign)
```

```
[1] 35
```

Hide

```
rep(globalAlign)
```

```
Global PairwiseAlignmentsSingleSubject (1 of 1)
pattern: ACTTCACCAGCTCCCTGGCGGTAAGTTGATCAAAGGAAAC------
subject: TTT----CGGGTAAGTAAATATATGTT--TCACTACTTCCTTTCGG
score: -85
```

Hide

```
score(globalAlign) # alignment score
```

```
[1] -85
```

Hide

```
type(globalAlign) # alignment type
```

```
[1] "global"
```

Hide

```
DNA_ALPHABET # show full nucleotide alphabet
```

```
 [1] "A" "C" "G" "T" "M" "R" "W" "S" "Y" "K" "V" "H" "D" "B" "N" "-" "+"
[18] "."
```

Hide

```
N <- 1000 # number of desired sequences

# strings have 0-36 characters from the adapters attached to each end
adapter <- DNAString("GATCGGAAGAGCTCGTATGCCGTCTTCTGCTTGAAA")
adapter
```

```
36-letter DNAString object
seq: GATCGGAAGAGCTCGTATGCCGTCTTCTGCTTGAAA
```

Hide

```
set.seed(123)
# used for function input
experiment <- list(side = rbinom(N,1,0.5),
                   width = sample(0:36,N,replace = TRUE))
```

```
# 2.3 | SEQUENCE ALIGNMENT SUMMARY

#Functions related to alignment summary

#summary alphabet() compareStrings()
#deletion() mismatchTable()
#nchar() nedit() indel()
#insertion() nindel()
#nmatch() nmismatch()
#pattern() subject()
#pid() rep() score() type()
```

Hide

```
# ''' Function to Generate DNA sequences /w these fragments '''
# The following code simulates what sequences with adapter fragments at either end could look li
ke during an experiment
# https://www.bioconductor.org/packages/devel/bioc/vignettes/Biostrings/inst/doc/PairwiseAlignme
nts.pdf

simulateReads <-
function(N, adapter, experiment, substitutionRate = 0.01, gapRate = 0.001) {

    chars <- strsplit(as.character(adapter), "")[[1]]
    sapply(seq_len(N), function(i, experiment, substitutionRate, gapRate) {

        width <- experiment[["width"]][i]
        side <- experiment[["side"]][i]
        randomLetters <- function(n) sample(DNA_ALPHABET[1:4], n, replace = TRUE)

        randomLettersWithEmpty <- function(n)
            sample(c("", DNA_ALPHABET[1:4]), n, replace = TRUE,
                   prob = c(1 - gapRate, rep(gapRate/4, 4)))

        nChars <- length(chars)
        value <- paste(ifelse(rbinom(nChars,1,substitutionRate),
                              randomLetters(nChars), chars),
                       randomLettersWithEmpty(nChars),sep = "", collapse = "")
        if (side)
            value <- paste(c(randomLetters(36 - width),
                             substring(value, 1, width)),
                           sep = "", collapse = "")
        else
            value <- paste(c(substring(value, 37 - width, 36),
                             randomLetters(36 - width)),
                           sep = "", collapse = "")
        value }, experiment = experiment, substitutionRate = substitutionRate, gapRate = gapRat
e)
}
```

Hide

```
# Generate Sequences w/ adapters from predefined function
adapterStrings <- simulateReads(N,
                                adapter,
                                experiment,
                                substitutionRate = 0.01,
                                gapRate = 0.001)

# 1000 sequences of 36 signal length intervals
adapterStrings <- DNAStringSet(adapterStrings)
adapterStrings # strings that contain adapters
```

```
DNAStringSet object of length 1000:
       width seq
   [1]    36 TTCTGCTTGAAAGTTCGCGAGAACAACTAGTCCGCA
   [2]    36 ATAACTACACTGGGTAACACAAACCTTTGGATCGGA
   [3]    36 AAGTGCGGTAGATGCTCTGAATGCTAGCCCGTCGCA
   [4]    36 TGGACGTGCGAATGCCAAATTGTAAGCGCGGGATCG
   [5]    36 ACCTGCAGAGTACGGATCGGAAGAGCTCGTATGCCG
   ...   ... ...
 [996]    36 TCCCTGACACGATAGATAACTCATTAGATTGGATCG
 [997]    36 TCAGGTGATGAAAGCATCTTTGGATCGGAAGAGCTC
 [998]    36 CGGAAGAGCTCGTATGCCGTCTTCTGCTTGAAAAGC
 [999]    36 ACGATCGGAAGAGCTCGTATGCCGTCTTGTGCTTGA
[1000]    36 TGCTTGAAATAAAGACTACACAGCAGCTGCAGTATT
```

Hide

```
# Generate Random DNA samples

M <- 5000
samples <- sample(DNA_ALPHABET[1:4], #Only 4 main nucleotides
                  36*M,
                  replace = TRUE)

typeof(samples) #check type
```

```
[1] "character"
```

Hide

```
#generate matrix of samples
sample_mat <- matrix(samples, nrow = M)
typeof(sample_mat)
```

```
[1] "character"
```

Hide

```
randomStrings <- apply(sample_mat, 1, paste, collapse = "")

randomStrings<- DNAStringSet(randomStrings)
randomStrings
```

```
DNAStringSet object of length 5000:
       width seq
   [1]    36 TAGTTATAAGCGGTCTCCTTTGCCAGATGAAAAATA
   [2]    36 ACAATCCGAGTTGTTTGCTCGGAGAGAATGCCGTCC
   [3]    36 AATATAACAGTCGTTTTGACCTATGTGCTACCGTTA
   [4]    36 ACAGTTGAAACAATCATAGGACGGGGAGTGTGTATT
   [5]    36 TCAATAACGATTCTTTTTCCATCAGTCTACAGATGC
   ...   ... ...
[4996]    36 CCCGTATTCGCGATCGGCAGCTCGTGGACACGGAGG
[4997]    36 GCGAGTGCTGTCGCCAGCATGCGCAACATTTTCAAT
[4998]    36 TAGGCTGTCGGAAGATAAGCCTCGCCATCGTGCCAT
[4999]    36 TTACGATCGTTCAGTCGATTATAACGGCACGCATCA
[5000]    36 CCTCCGTCGAGTCACCTGTTGAAACTATATGAGAAT
```

2.4 | SEQUENCE ALIGNMENT APPLICATION

REMOVING ADAPTERS FROM SEQUENCE READINGS An interesting PSA example was shown in the Pairwise Sequence Reference & is related to experimentally processed DNA sequences Trimming adapter sequences - is it necessary?

Removal of adapter sequences in a process called read trimming, or clipping, is one of the first steps in analyzing NGS data. With more than 30 published adapter trimming tools there is a more than large choice for the appropriate tool. Yet, there is a debate whether this step really is as important as the number of tools suggests, or whether it is possible to skip this time-consuming step for many NGS applications.

Finding and removing uninteresting experiment process-related fragments like adapters is a common problem in genetic sequencing Pairwise Sequence Alignment is well suited to address this sort of issue, as this problem relates to sequence similarity When adapters are used to anchor or extend a sequence during the experiment process, they either intentionally or unintentionally become sequenced during the read process & thus are present in the sequence

Hide

```
#Substitution MAtrix
submat1 <- nucleotideSubstitutionMatrix(match =0, mismatch = -1, baseOnly =
                                          TRUE)
# adapter strings DNA & adapter (0-36 characters attached to either end)
# should have higher hit rate

adapterAligns1 <- pairwiseAlignment(adapterStrings,
                                    adapter,
                                    substitutionMatrix = submat1,
                                    gapOpening = 0, gapExtension = 1)

adapterAligns1 # PairwiseAlignmentsSingleSubject (contains multiple PSA)]
```

```
Global PairwiseAlignmentsSingleSubject (1 of 1000)
pattern: TTCTGCTTGAA-AGTTCGCGAGAACAACTAGTCC--GCA-
subject: GA-T-CG-GAAGAGCTCGTATGC-CGTCTTCTGCTTGAAA
score: -22
```

Hide

```
adapterAligns1_score <- score(adapterAligns1)
```

Hide

```
# random DNA & adapter (baseline for comparison only)
randomScores1 <- pairwiseAlignment(randomStrings,
                                   adapter,
                                   substitutionMatrix = submat1,
                                   gapOpening = 0, gapExtension = 1,
                                   scoreOnly = TRUE) # get the final alignment score only
```

Hide

```
# show the quantile data 99%+ score
quantile(randomScores1, seq(0.99,1,0.001))
```

```
  99% 99.1% 99.2% 99.3% 99.4% 99.5% 99.6% 99.7% 99.8% 99.9%  100% 
  -16   -16   -16   -16   -16   -16   -16   -16   -15   -15   -14 
```

Using completely random strings as a baseline for any PSA methodology we develop to remove the adapter characters So let's create randomised DNA sequences using the DNA_ALPHABET using sample()

Hide

```
# find places where the adapter scores are higher than in baseline (using onlu 99.9% quartile da
ta only)
# 29th character +=
table(adapterAligns1_score > quantile(randomScores1,0.999), experiment[["width"]])
```

```
       0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
  FALSE 18 26 21 17 31 25 27 29 30 30 37 26 29 25 30 27 32 29 36 16 23
  TRUE   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

       21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
  FALSE 23 32 27 31 24 25 28 31  4  0  0  0  0  0  0  0
  TRUE   0  0  0  0  0  0  0  0 23 26 25 28 25 34 23 27
```

METHOD 1 For the first approach, we'll use a match/mismatch of 0/-1 for the substitution matrix gap opening of 0 & gapEXtension of 1

Hide

```
# [1] read clustaw format (.aln)
origMAlign <- readDNAMultipleAlignment(filepath = system.file("extdata","msx2_mRNA.aln",
                                                              package="Biostrings"),
                                                              format="clustal")
# [1] read phylip format (.txt)
phylipMAlign <- readAAMultipleAlignment(filepath = system.file("extdata","Phylip.txt",
                                                               package="Biostrings"),
                                                               format="phylip")
```

Hide

```
origMAlign
```

```
DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                              names
[1] -----TCCCGTCTCCGCAGCAA...AATTAAAAAAAAAAAAAAAAA gi|84452153|ref|N...
[2] ----------------------...---------------------- gi|208431713|ref|...
[3] ----------------------...---------------------- gi|118601823|ref|...
[4] ----------------------...---------------------- gi|114326503|ref|...
[5] ----------------------...---------------------- gi|119220589|ref|...
[6] ----------------------...---------------------- gi|148540149|ref|...
[7] --------------CGGCTCCG...---------------------- gi|45383056|ref|N...
[8] GGGGGAGACTTCAGAAGTTGTT...---------------------- gi|213515133|ref|...
```

Hide

```
DNAStr = as(origMAlign, "DNAStringSet") #change DNAMultipleAlignment ->DNAStringset

#Write to files
writeXStringSet(DNAStr, file="DNAStr.fasta" ) #write in fasta format

write.phylip(phylipMAlign, filepath = "phylipMalign.txt") #write in Phylip format
```

Hide

```
#Display an alignment
origMAlign
```

```
DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                              names
[1] -----TCCCGTCTCCGCAGCAA...AATTAAAAAAAAAAAAAAAAA gi|84452153|ref|N...
[2] ----------------------...---------------------- gi|208431713|ref|...
[3] ----------------------...---------------------- gi|118601823|ref|...
[4] ----------------------...---------------------- gi|114326503|ref|...
[5] ----------------------...---------------------- gi|119220589|ref|...
[6] ----------------------...---------------------- gi|148540149|ref|...
[7] --------------CGGCTCCG...---------------------- gi|45383056|ref|N...
[8] GGGGGAGACTTCAGAAGTTGTT...---------------------- gi|213515133|ref|...
```

3 | ALIGNMENT OBJECTS Quite a number of application in Bioinformatics involve the use of biological sequence alignment We can read an alignment file using readDNAMultipleAlignment(filepath), examples shown below Masking is also used for various operations surrounding sequence alignments, in particular when we have lots of gaps in our alignments & want to remove them before using the data for analysis 3.1 | IO ALIGNMENT

READ ALIGNMENT Read Alignment | Two formats used for alignment: clustal, phylip

Hide

```
#display alignment
phylipMAlign
```

```
AAMultipleAlignment with 24 rows and 181 columns
      aln                                          names
 [1] YVID-QMISAKAIAARVEALG...GLDYAQNHRNLPFIGTVRFTD hprt_rhoca
 [2] HHVD-VLISENDVHARIAELG...GIDYAQRHRNLGYIGKVVLEE hprt_haein
 [3] HHVD-VLISENDVHARIAELG...GIDYAQRHRNLGYIGKVVLEE hprt_haein
 [4] HTVE-VMISEQEVQERIRELG...GIDYAQKYRDLPFIGKVVPQE hprt_vibha
 [5] HTVE-VMIPEAEIKARIAELG...GIDYAQRYRHLPYIGKVILLD hprt_ecoli
 [6] EDLEKVFIPHGLIMDRTERLA...ALDYNEYFRDLNHVCVISESG hprt_merun
 [7] EDLERVFIPHGLIMDRTERLA...ALDYNEYFRDLNHVCVISETG hprt_monke
 [8] EDLERVFIPHGLIMDRTERLA...ALDYNEYFRDLNHVCVISETG hprt_human
 [9] EDLEKVFIPHGLIMDRTERLA...ALDYNEHFRDLNHVCVISESG hprt_rat
 ... ...
[16] DVLESLLATFEECKALAADTA...GLDDNGLRRGWAHLFDINLSE gprt_giard
[17] DFATSVLFTEAELHTRMRGVA...GLDYDQSYREVRDVVILKPSV hprt_trybb
[18] EFAEKILFTEEEIRTRIMEVA...GLDYDDTYRELRDIVVLRPEV hprt_tcruz
[19] PMSAHTLVTQEQVWAATAKCA...GMDYAESYRELRDICVLKKEY hprt_leido
[20] PMSCRTLATQEQIWSATAKCA...GMDFAEAYRELRDVCVLKKEY hprt_crifa
[21] DDLERVLYNQDDIQKRIRELA...GFDFHNKYRNLPVIGILKESV hgxr_trifp
[22] KAIEKVLVSEEEIIEKSKELG...GLDYEENYRNLPYVGVLKPEV hprt_lacla
[23] HDIEKVLISEEEIQKKVKELG...GLDYAERYRNLPYIGVLKPAV hprt_bacsu
[24] MGIKSIVINEQQIEEGCQKAV...GLDYDGFYRNLPYVGVFEPDN hprt_mycge
```

WRITING ALIGNMENT TO FILE We can write alignments using two different formats; FASTA & Phylip formats

Hide

```
rownames (origMAlign) #show all
```

```
[1] "gi|84452153|ref|NM_002449.4|"   "gi|208431713|ref|NM_001135625."
[3] "gi|118601823|ref|NM_001079614." "gi|114326503|ref|NM_013601.2|"
[5] "gi|119220589|ref|NM_012982.3|"  "gi|148540149|ref|NM_001003098."
[7] "gi|45383056|ref|NM_204559.1|"   "gi|213515133|ref|NM_001141603."
```

Hide

```
rownames(origMAlign)[1]  #show just the one
```

```
[1] "gi|84452153|ref|NM_002449.4|"
```

Hide

```
# [3] Make our own list of names & assign it to alignment rownames
# These names are more are more easily interpretable
rownames(origMAlign) <- c("Human","Chimp","Cow","Mouse","Rat","Dog","Chicken","Salmon") # concat
characters
origMAlign
```

```
DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                             names
[1] -----TCCCGTCTCCGCAGCAA...AATTAAAAAAAAAAAAAAAAA Human
[2] ----------------------...---------------------- Chimp
[3] ----------------------...---------------------- Cow
[4] ----------------------...---------------------- Mouse
[5] ----------------------...---------------------- Rat
[6] ----------------------...---------------------- Dog
[7] --------------CGGCTCCG...---------------------- Chicken
[8] GGGGGAGACTTCAGAAGTTGTT...---------------------- Salmon
```

DISPLAY ALIGNMENT We can display the alignment via the object instance & the get the corresponding individual alignment name using rownames

Hide

```
# [4] Detail provides a view for all of the alignment
 detail(origMAlign)
```

Hide

```
# [5] We can set rowmask w/ IRanges to hide some rows in alignment
# let's mask the first three rows

Test <- origMAlign
rowmask(Test) <- IRanges(start=1,end=3) # set int range function
Test
```

```
DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                             names
[1] ######################...###################### Human
[2] ######################...###################### Chimp
[3] ######################...###################### Cow
[4] ----------------------...---------------------- Mouse
[5] ----------------------...---------------------- Rat
[6] ----------------------...---------------------- Dog
[7] --------------CGGCTCCG...---------------------- Chicken
[8] GGGGGAGACTTCAGAAGTTGTT...---------------------- Salmon
```

Hide

```
# remove rowmask
rowmask(Test) <- NULL
```

Hide

```
# [6] We can also use column masking
# concat can be used to select multiple locations
# let's mask the columns -> 1-500 & 1000-2343

Test <- origMAlign
colmask(Test) <- IRanges(2,4)
colmask(Test) <- IRanges(6,8) # You can add multiple masks as well
Test
```

```
DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                               names
[1] -###-###CGTCTCCGCAGCAA...AATTAAAAAAAAAAAAAAAAA Human
[2] -###-###--------------...---------------------- Chimp
[3] -###-###--------------...---------------------- Cow
[4] -###-###--------------...---------------------- Mouse
[5] -###-###--------------...---------------------- Rat
[6] -###-###--------------...---------------------- Dog
[7] -###-###------CGGCTCCG...---------------------- Chicken
[8] G###G###CTTCAGAAGTTGTT...---------------------- Salmon
```

Hide

```
# remove column mask
colmask(Test) <- NULL
```

CHANGE ALIGNMENT NAMES Set Alignment Names | rownames(aln) - Replace alignment names if we need to make it more clear for interpretation

Hide

```
origMAlign
```

```
DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                               names
[1] -----TCCCGTCTCCGCAGCAA...AATTAAAAAAAAAAAAAAAAA Human
[2] ----------------------...---------------------- Chimp
[3] ----------------------...---------------------- Cow
[4] ----------------------...---------------------- Mouse
[5] ----------------------...---------------------- Rat
[6] ----------------------...---------------------- Dog
[7] --------------CGGCTCCG...---------------------- Chicken
[8] GGGGGAGACTTCAGAAGTTGTT...---------------------- Salmon
```

SHOW DETAILED ALIGNMENT Show entire alignment | detail(aln) - can be used to display the entire sequence alignment

Hide

```
#a mask was found @1232 - 1236 of first row

tata_mask <- maskMotif(origMAlign, "AAAA")
colmask(tata_mask)
```

```
NormalIRanges object with 3 ranges and 0 metadata columns:
          start       end     width
      <integer> <integer> <integer>
  [1]       666       669         4
  [2]      1200      1203         4
  [3]      1232      1236         5
```

3.2 | ALIGNMENT MASKING

We'll look at several types of alignment masking; basic masking, motif masking & gap masking

BASIC MASKING Hiding Rows | rowmask(aln) - used for hiding some of the row content in an alignment Hiding Columns | colmask(aln) - used for hiding some of the column content in an alignment

Hide

```
autoMasked <- maskGaps(origMAlign, min.fraction = 0.5, min.block.width =4)

autoMasked
```

```
DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                             names
[1] ######################...###################### Human
[2] ######################...###################### Chimp
[3] ######################...###################### Cow
[4] ######################...###################### Mouse
[5] ######################...###################### Rat
[6] ######################...###################### Dog
[7] ######################...###################### Chicken
[8] ######################...###################### Salmon
```

Hide

```
# Multiple sequence alignment in matrix format
full = as.matrix(origMAlign)
dim(full)
```

```
[1]    8 2343
```

MOTIF MASKING Masking with Motifs | Useful for masking subsequence occurences of a string from columns where it is present in the consensus sequence

Hide

```
#if we mask the entire row, we get NA
Test <- origMAlign

rowmask(Test) <- IRanges(start = 1, end = 3) #set int range function

alphabetFrequency(Test)
```

```
      A   C   G   T  M  R  W  S  Y  K  V  H  D  B  N    -  +  .
[1,]  NA  NA  NA  NA NA NA NA NA NA NA NA NA NA NA NA   NA NA NA
[2,]  NA  NA  NA  NA NA NA NA NA NA NA NA NA NA NA NA   NA NA NA
[3,]  NA  NA  NA  NA NA NA NA NA NA NA NA NA NA NA NA   NA NA NA
[4,] 538 519 501 604  0  0  0  0  0  0  0  0  0  0  0  181  0  0
[5,] 494 483 477 522  0  0  0  0  0  0  0  0  0  0  0  367  0  0
[6,] 160 285 241 118  0  0  0  0  0  0  0  0  0  0  0 1539  0  0
[7,] 235 376 300 196  0  0  0  0  0  0  0  0  0  0  0 1236  0  0
[8,] 311 326 314 321  0  0  0  0  0  0  0  0  0  0  0 1071  0  0
```

Hide

```
# [1] If we masked only parts of the row content, we'll get freq of only those that aren't maske
d
autoMasked <- maskGaps(origMAlign,
                       min.fraction=0.5,
                       min.block.width=4)
alphabetFrequency(autoMasked)
```

```
      A   C   G   T M R W S Y K V H D B N   - + .
[1,] 260 351 296 218 0 0 0 0 0 0 0 0 0 0 0  18 0 0
[2,] 171 271 231 128 0 0 0 0 0 0 0 0 0 0 3 339 0 0
[3,] 277 360 275 209 0 0 0 0 0 0 0 0 0 0 0  22 0 0
[4,] 265 343 277 226 0 0 0 0 0 0 0 0 0 0 0  32 0 0
[5,] 251 345 287 229 0 0 0 0 0 0 0 0 0 0 0  31 0 0
[6,] 160 285 241 118 0 0 0 0 0 0 0 0 0 0 0 339 0 0
[7,] 224 342 273 190 0 0 0 0 0 0 0 0 0 0 0 114 0 0
[8,] 268 289 273 262 0 0 0 0 0 0 0 0 0 0 0  51 0 0
```

GAP MASKING Masking alignments with gaps | Useful for when we need to mask gaps that are present in the alignment

MaskGaps also operate on columns & will mask columns based on the fraction of each column that contains gaps; min.fraction along with the width of columns that contain this fraction of gaps min.block.width

Hide

```
# ''' Bad Cluster Case '''

# Calculate the distance to eachother (alignments)

str_set <- as(origMAlign, "DNAStringSet") #convert/use alignment to/as string set

class(str_set) #DNAStringSet
```

```
[1] "DNAStringSet"
attr(,"package")
[1] "Biostrings"
```

Hide

```
str_set #the stringset only contains those present in the mask
```

```
DNAStringSet object of length 8:
    width seq                                          names
[1]  2343 -----TCCCGTCTCCGCAG...TTAAAAAAAAAAAAAAAAA Human
[2]  2343 --------------------...-------------------- Chimp
[3]  2343 --------------------...-------------------- Cow
[4]  2343 --------------------...-------------------- Mouse
[5]  2343 --------------------...-------------------- Rat
[6]  2343 --------------------...-------------------- Dog
[7]  2343 --------------CGGCT...-------------------- Chicken
[8]  2343 GGGGGAGACTTCAGAAGTT...-------------------- Salmon
```

Hide

```
#Calculate Distance
sdist <- stringDist(str_set, method = 'hamming')

sdist
```

```
        Human Chimp  Cow Mouse  Rat  Dog Chicken
Chimp    1424
Cow      1225   382
Mouse     772  1457 1257
Rat       783  1267 1080   431
Dog      1497    79  392  1463 1276
Chicken  1504   514  524  1489 1379  526
Salmon   1691   904  808  1651 1550  916     816
```

Hide

```
# cluster using Hierarchical clustering, hclust
clust <- hclust(sdist,
                method = "single")
clust
```

```
Call:
hclust(d = sdist, method = "single")

Cluster method   : single
Distance         : hamming
Number of objects: 8
```

Hide

```
pdf(file="tree1.pdf") # plot the clustering
plot(clust) # plot dendogram of the clustering
dev.off()
```

```
null device
          1
```

Hide

```
# Cut the tree into four groups
fourgroups <- cutree(clust, 4)
fourgroups
```

```
  Human   Chimp     Cow   Mouse     Rat     Dog Chicken  Salmon
      1       2       2       3       3       2       2       4
```

Hide

```
# ''' Better Cluster Case '''

# suppose we have created some mask for our alignment
autoMasked
```

```
DNAMultipleAlignment with 8 rows and 2343 columns
     aln                                                 names
[1] ######################...###################### Human
[2] ######################...###################### Chimp
[3] ######################...###################### Cow
[4] ######################...###################### Mouse
[5] ######################...###################### Rat
[6] ######################...###################### Dog
[7] ######################...###################### Chicken
[8] ######################...###################### Salmon
```

Hide

```
# Calculate the distance to eachother (alignments)
class(autoMasked) # DNAMultipleAlignment class
```

```
[1] "DNAMultipleAlignment"
attr(,"package")
[1] "Biostrings"
```

Hide

```
str_set <- as(autoMasked,"DNAStringSet") # convert/use alignment to/as string set
class(str_set) # DNAStringSet
```

```
[1] "DNAStringSet"
attr(,"package")
[1] "Biostrings"
```

Hide

```
str_set # the stringset only contains those present in the mask
```

```
DNAStringSet object of length 8:
    width seq                                          names
[1]  1143 CAGAGAAGTCA-TGGCTTC...AGCAGACGTAAAAATTCAA Human
[2]  1143 ----------A-TGGCTTC...------------------- Chimp
[3]  1143 GAGAGAAGTCA-TGGCTTC...AGCAAAAAAAAAAAAAAAA Cow
[4]  1143 CAGA-AAGTCA-TGGCTTC...GCCAGATGTAAAAATTCAA Mouse
[5]  1143 ----------A-TGGCTTC...GCCAGATGTAAAAATTCAA Rat
[6]  1143 ----------A-TGGCTTC...------------------- Dog
[7]  1143 CGGCCCCGCTC-CAGCCAC...------------------- Chicken
[8]  1143 TGTGTTCGTCAACATCTGA...ATTTATTCTATAGCCCTGA Salmon
```

Hide

```
# Calculate distance
sdist <- stringDist(str_set,
                    method="hamming")
sdist
```

```
        Human Chimp Cow Mouse Rat Dog Chicken
Chimp     325
Cow       130   378
Mouse     178   406 202
Rat       186   403 212    77
Dog       398    79 388   412 412
Chicken   422   436 442   439 437 448
Salmon    625   724 630   619 616 736     639
```

Hide

```
# cluster using Hierarchical clustering, hclust
clust <- hclust(sdist,
                method = "single")
clust
```

```

Call:
hclust(d = sdist, method = "single")

Cluster method   : single
Distance         : hamming
Number of objects: 8
```

Hide

```
pdf(file="tree2.pdf") # plot the clustering
plot(clust) # plot dendogram of the clustering
dev.off()
```

```
null device
          1
```

Hide

```
# Cut the tree into four groups
fourgroups <- cutree(clust, 4)
fourgroups
```

```
  Human   Chimp     Cow   Mouse     Rat     Dog Chicken  Salmon
      1       2       1       1       1       2       3       4
```

3.3 | ALIGNMENT MASKING APPLICATIONS

ALPHABET FREQUENCY w/ MASKING Having created masks for parts of the alignment which is of interest to us, we can conduct some form of investigation When using masks, operations will only include the non masked sequence characters, eg. alphabetFrequency.

Hide

```
suppressPackageStartupMessages(library(msa))
```

Hide

```
# Load Example File
mySequenceFile <- system.file("examples",
                              "exampleAA.fasta",
                              package="msa")

# Read Amino acid string set
mySequences <- readAAStringSet(mySequenceFile) # read stringset (same as biostrings library)
mySequences
```

```
AAStringSet object of length 9:
    width seq                                          names
[1]   452 MSTAVLENPGLGRKLSDFG...ADSINSEIGILCSALQKIK PH4H_Homo_sapiens
[2]   453 MAAVVLENGVLSRKLSDFG...DSINSEVGILCNALQKIKS PH4H_Rattus_norve...
[3]   453 MAAVVLENGVLSRKLSDFG...DSINSEVGILCHALQKIKS PH4H_Mus_musculus
[4]   297 MNDRADFVVPDITTRKNVG...DDLVLNAGDRQGWADTEDV PH4H_Chromobacter...
[5]   262 MKTTQYVARQPDDNGFIHY...HEAMRLGLHAPLFPPKQAA PH4H_Pseudomonas_...
[6]   451 MSALVLESRALGRKLSDFG...ADSISSEVEILCSALQKLK PH4H_Bos_taurus
[7]   313 MAIATPTSAAPTPAPAGFT...GDAVLNAGTREGWADTADI PH4H_Ralstonia_so...
[8]   294 MSGDGLSNGPPPGARPDWT...RGTQAYATAGGRLAGAAAG PH4H_Caulobacter_...
[9]   275 MSVAEYARDCAAQGLRGDY...FEAIVARRKDQKALDPATV PH4H_Rhizobium_loti
```

SEQUENCE SET CLUSTERING w/ MASKING We can also cluster the alignments in a StringSet based on their distance (stringDist) to each other | hclust Passing a DNAStringSet, the clustering will also take into account only those alphabet in the created masking | String Distance & Clustering Video Here we'll look at two cases, unmasked alignments & masked alginments, the benefit of masking being that the alignments contain lots of gaps (origMAlign)

Hide

```
#Multiple Sequence Alignment
aln <- msa(mySequences) #ClustalW used by default
```

```
use default substitution matrix
```

Hide

```
#same masking used in biostrings can be used

rowmask(aln, invert= TRUE) <- IRanges(start = 1, end = 3)
#print (aln, show= "complete") #show full alignment

print(aln)
```

```
CLUSTAL 2.1

Call:
   msa(mySequences)

MsaAAMultipleAlignment with 9 rows and 456 columns
    aln                                              names
[1] MAAVVLENGVLSRKLSDFGQET...LADSINSEVGILCNALQKIKS PH4H_Rattus_norve...
[2] MAAVVLENGVLSRKLSDFGQET...LADSINSEVGILCHALQKIKS PH4H_Mus_musculus
[3] MSTAVLENPGLGRKLSDFGQET...LADSINSEIGILCSALQKIK- PH4H_Homo_sapiens
[4] ######################...###################### PH4H_Bos_taurus
[5] ######################...###################### PH4H_Chromobacter...
[6] ######################...###################### PH4H_Ralstonia_so...
[7] ######################...###################### PH4H_Caulobacter_...
[8] ######################...###################### PH4H_Pseudomonas_...
[9] ######################...###################### PH4H_Rhizobium_loti
Con MAAVVLENGVLSRKLSDFGQET...LADSINSEVGILC?ALQKIKS Consensus
```

Hide

```
#MSA approach options
myClustalWAlignment <- msa(mySequences, "ClustalW")
```

```
use default substitution matrix
```

Hide

```
myClustalOmegaAlignment <- msa(mySequences, "ClustalOmega")
```

```
using Gonnet
```

Hide

```
myMuscleAlignment <- msa(mySequences, "Muscle")
```

BIOCONDUCTOR :: msa The method used in biological sequence alignment can’t handle lots of alignments described in snipplet: Most alignments are computed using the progressive alignment heuristic These methods are starting to become a bottleneck in some analysis pipelines when faced with data sets of the size of many thousands of sequences CLUSTALW, CLUSTALOMEGA, MUSCLE are all more advanced methods of multiple sequence alignment, varying in algorithm, but achieving the same goal So for realistic problems, we may have to compare lots of sequences togther, thus the above three algorithms are more preferable, to keep computational cost low Upon msa, we get MsaAAMultipleAlignment objects, which we already used in Section 3; the same alignment related operations used in Biostrings can be used (eg. masking)

Hide

```
# using as() to change msa alignment type to StringSet
AAStr = as(myMuscleAlignment, "AAStringSet") # output as String Set
writeXStringSet(AAStr, file="AAStr.fasta") # write in FASTA format
```

Hide

```
# Load Example File
mySequenceFile <- system.file("examples",
                              "exampleAA.fasta",
                              package="msa")

# Read Amino acid string set
mySequences <- readAAStringSet(mySequenceFile) # read stringset (same as biostrings library)
mySequences
```

Hide

```
#Multiple Sequence Alignment
aln <- msa(mySequences) #ClustalW used by default

#same masking used in biostrings can be used

rowmask(aln, invert= TRUE) <- IRanges(start = 1, end = 3)
#print (aln, show= "complete") #show full alignment

print(aln)
```

Hide

```
#MSA approach options
myClustalWAlignment <- msa(mySequences, "ClustalW")
myClustalOmegaAlignment <- msa(mySequences, "ClustalOmega")
myMuscleAlignment <- msa(mySequences, "Muscle")
```

Hide

```
# using as() to change msa alignment type to StringSet
AAStr = as(myMuscleAlignment, "AAStringSet") # output as String Set
writeXStringSet(AAStr, file="AAStr.fasta") # write in FASTA format
```

Project Files & template from Andrey Shtrauss