Recap From Last Time:

- **Substitution matrices**: Where our alignment match and mis-match scores typically come from
- **Comparing methods**: The trade-off between *sensitivity*, *selectivity* and *performance*
- **Sequence motifs and patterns**: Finding functional cues from conservation patterns
- **Sequence profiles and position specific scoring matrices (PSSMs)**, Building and searching with profiles, Their advantages and limitations
- **PSI-BLAST algorithm**: Application of iterative PSSM searching to improve BLAST sensitivity
- **Hidden Markov models (HMMs)**: More versatile probabilistic model for detection of remote similarities
Today’s Learning Goals

• Familiarity with R’s basic syntax.

• Familiarity with major R data structures.

• Understand the basics of using functions.

• Be able to use R to read and parse comma-separated (.csv) formatted files ready for subsequent analysis.

• Appreciate how you can use R scripts to aid with reproducibility.
What is R?

R is a freely distributed and widely used programming language and environment for statistical computing, data analysis and graphics.

R provides an unparalleled interactive environment for data analysis.

It is script-based (i.e. driven by computer code) and not GUI-based (point and click with menus).
What is R?

R is a freely distributed and widely used programming language and environment for statistical computing, data analysis and graphics. R provides an unparalleled interactive environment for data analysis. It is script-based (i.e. driven by computer code) and not GUI-based (point and click with menus).

R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under certain conditions. Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors. Type 'contributors()' for more information and 'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help. Type 'q()' to quit R.
What is R?

R is a freely distributed and widely used programing language and environment for statistical computing, data analysis and graphics.

R provides an unparalleled interactive environment for data analysis. It is script-based (i.e. driven by computer code) and not GUI-based (point and click with menus).

Type "R" in your terminal.
What is R?

R is a freely distributed and widely used programing language and environment for statistical computing, data analysis and graphics.

R provides an unparalleled interactive environment for data analysis. It is script-based (i.e. driven by computer code) and not GUI-based (point and click with menus).

Type "R" in your terminal

This is the R prompt
R is a freely distributed and widely used programming language and environment for statistical computing, data analysis, and graphics. R provides an unparalleled interactive environment for data analysis. It is script-based (i.e. driven by computer code) and not GUI-based (point and click with menus).

Type "R" in your terminal

This is the R prompt: Type `q()` to quit!
What R is NOT

A performance optimized software library for incorporation into your own C/C++ etc. programs.

A molecular graphics program with a slick GUI.

Backed by a commercial guarantee or license.

Microsoft Excel!
What about Excel?

- Data manipulation is easy
- Can see what is happening
- **But**: graphics are poor
- Looping is hard
- Limited statistical capabilities
- Inflexible and irreproducible
- There are many many things Excel just cannot do!

Use the right tool!
**Rule of thumb**: Every analysis you do on a dataset will have to be redone 10–15 times before publication. Plan accordingly!
Why use R?

- Productivity
- Flexibility
- Designed for data analysis
### IEEE 2016 Top Programming Languages

<table>
<thead>
<tr>
<th>Language Rank</th>
<th>Types</th>
<th>Spectrum Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. C</td>
<td>📱💻📱</td>
<td>100.0</td>
</tr>
<tr>
<td>2. Java</td>
<td>🌍📱💻</td>
<td>98.1</td>
</tr>
<tr>
<td>3. Python</td>
<td>🌍💻</td>
<td>98.0</td>
</tr>
<tr>
<td>4. C++</td>
<td>📱💻📱</td>
<td>95.9</td>
</tr>
<tr>
<td>5. R</td>
<td>📱💻</td>
<td>87.9</td>
</tr>
<tr>
<td>6. C#</td>
<td>🌍📱💻</td>
<td>86.7</td>
</tr>
<tr>
<td>7. PHP</td>
<td>🌍💻</td>
<td>82.8</td>
</tr>
<tr>
<td>8. JavaScript</td>
<td>🌍📱</td>
<td>82.2</td>
</tr>
<tr>
<td>9. Ruby</td>
<td>🌍💻</td>
<td>74.5</td>
</tr>
<tr>
<td>10. Go</td>
<td>🌍💻</td>
<td>71.9</td>
</tr>
</tbody>
</table>

http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages
R and Python: The Numbers

Popularity Rankings

R and Python's popularity between 2013 and February 2015 (Flobes's Index)

Python

Redmonk ranking, comparing the relative performance of programming languages on GitHub and Stack Overflow (September 2012 and January 2013, 2014, 2015)

Jobs And Salary?

2014 Dice Tech Salary Survey:
Average Salary For High Paying Skills and Experience

R

$115,531

Python

$94,139

• R is the “lingua franca” of data science in industry and academia.

• Large user and developer community.
  • As of April 13th 2018 there are 12,481 add on R packages on CRAN and 1,473 on Bioconductor - more on these later!

• Virtually every statistical technique is either already built into R, or available as a free package.

• Unparalleled exploratory data analysis environment.
<table>
<thead>
<tr>
<th>Modularity</th>
<th>Core R functions are modular and work well with others</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactivity</td>
<td>R offers an unparalleled exploratory data analysis environment</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>Access to existing tools and cutting-edge statistical and graphical methods</td>
</tr>
<tr>
<td>Support</td>
<td>Extensive documentation and tutorials available online for R</td>
</tr>
<tr>
<td>R Philosophy</td>
<td>Encourages open standards and reproducibility</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Modularity</strong></td>
<td>Core R functions are modular and work well with others</td>
</tr>
<tr>
<td><strong>Interactivity</strong></td>
<td>R offers an unparalleled exploratory data analysis environment</td>
</tr>
<tr>
<td><strong>Infrastructure</strong></td>
<td>Access to existing tools and cutting-edge statistical and graphical methods</td>
</tr>
<tr>
<td><strong>Support</strong></td>
<td>Extensive documentation and tutorials available online for R</td>
</tr>
<tr>
<td><strong>R Philosophy</strong></td>
<td>Encourages open standards and reproducibility</td>
</tr>
</tbody>
</table>
Modularity

R was designed to allow users to interactively build complex workflows by interfacing smaller ‘modular’ functions together.

An alternative approach is to write a single complex program that takes raw data as input, and after hours of data processing, outputs publication figures and a final table of results.

get.seq() → hmmer() → pdbaln() → pdbfit() → pca() → plot()
Which would you prefer and why?

Modular vs Custom
The ‘monster approach’ is customized to a particular project but results in massive, fragile and difficult to modify (therefore inflexible, untransferable, and error prone) code.

With modular workflows, it’s easier to:

• **Spot errors** and figure out where they’re occurring by inspecting intermediate results.

• **Experiment** with alternative methods by swapping out components.

• **Tackle novel problems** by remixing existing modular tools.
Another common approach to bioinformatics data analysis is to write individual scripts in Perl/Python/Awk/C etc. to carry out each subsequent step of an analysis.

This can offer many advantages but can be challenging to make robustly modular and interactive.
Interactivity & exploratory data analysis

Learning R will give you the freedom to explore and experiment with your data.

“Data analysis, like experimentation, must be considered as a highly interactive, iterative process, whose actual steps are selected segments of a stubbily branching, tree-like pattern of possible actions”. [J. W. Tukey]
Interactivity & exploratory data analysis

Learning R will give you the freedom to explore and experiment with your data.

“Data analysis, like experimentation, must be considered as a highly interactive, iterative process, whose actual steps are selected segments of a stubbily branching, tree-like pattern of possible actions”. [J. W. Tukey]

Bioinformatics data is intrinsically high dimensional and frequently ‘messy’ requiring exploratory data analysis to find patterns - both those that indicate interesting biological signals or suggest potential problems.
Input sequence or structure
\[ \text{aa} = \text{get.seq}("lrx2\_A") \]

BLAST or HMMER search
\[ \text{hits} = \text{blast}(\text{aa}) \]

Download related structures
\[ \text{files} = \text{get.pdb}(\text{hits}) \]

Align structures
\[ \text{pdds} = \text{pdbaln}(\text{files}) \]

Input user specific PDBs

Input simulation trajectory

Ensemble NMA
\[ \text{modes} = \text{nma}(\text{pdds}) \]

Ensemble PCA & addtional sequence/structure analysis
\[ \text{pc} = \text{pca}(\text{pdds}) \]
\# \text{conserv()} \text{ rmsd()} \]
\# \text{rmsf()} \text{ dccm()} \text{ etc.} \]

Fluctuation analysis
\[ \text{plot}(\text{modes}) \]

Correlation networks
\[ \text{cna}(\text{ci}) \]

X-ray & MD PCA
\[ \text{plot}(\text{pc}) \]

Clustering
\[ \text{heatmap}(\text{modes}) \]

PCA vs NMA
\[ \text{rmsip}(\text{pc}, \text{modes}) \]
R Features = `functions()`
How do we use R?
Two main ways to use R

1. Terminal

2. RStudio

1- Code Editor

2- R Console

3- Workspace and History

4- Plots and files
We will use **RStudio** today

1. Code Editor
2. R Console
3. Workspace and History
4. Plots and files
Let's get started...

1. Code Editor
2. R Console
3. Workspace and History
4. Plots and files
Some simple R commands

1. > 2+2
   [1] 4

2. > 3^2
   [1] 9

3. > sqrt(25)
   [1] 5

4. > 2*(1+1)
   [1] 4

5. > 2*1+1
   [1] 3

6. > exp(1)
   [1] 2.718282

7. > log(2.718282)
   [1] 1

8. > log(10, base=10)
   [1] 1

9. > log(10 , base = 10)
   [1] 1

10. > x=1:50
    > plot(x, sin(x))
Does your plot look like this?

![Graph of \( \sin(x) \) for \( x = 1:50 \)]
plot(x, sin(x), typ="l", col="blue", lwd=3, xlab="x = 1:50")

Options: ?plot ?plot.default
Key point: You need to visualize your data!
Learning a new language is hard!
Error Messages

Sometimes the commands you enter will generate errors. Common beginner examples include:

• Incomplete brackets or quotes e.g.
  
  
  $((4+8)*20 \ <\text{enter}> \ +$

  This returns a + here, which means you need to enter the remaining bracket - R is waiting for you to finish your input. Press <ESC> to abandon this line if you don't want to fix it.

• Not separating arguments by commas e.g.

  plot(1:10 col="red")

• Typos including miss-spelling functions and using wrong type of brackets e.g.

  exp{4}
Your turn!

https://bioboot.github.io/bgg213_S18/class-material/04_rintro/

If you have done the introductory DataCamp course then feel free to jump to section #3 *Object Assignment*
Topics Covered:

- Calling Functions
- Getting help in R
- Vectors and vectorization
- Workspace and working directory
- RStudio projects

Use when finished
Topics Covered:

- Calling Functions
- Getting help in R
- Vectors and vectorization
- Workspace and working directory
- RStudio projects
Vectors

- Vectors are the most basic data structure in R
- All elements of a vector must be the same type

```
dbl_var <- c(1, 2.5, 4.5)
log_var <- c(TRUE, FALSE, T, F)
chr_var <- c("these are", "some", "strings")
```

- When you attempt to combine different types they will be coerced to the most flexible type.

```
var <- c(1, "G", "4", 0.05, TRUE)
```
Names

• You can name a vector in several ways:
  
  • When creating it:  
    \[
    x \leftarrow c(a = 1, b = 2, c = 3) 
    \]

  • By modifying an existing vector in place:  
    \[
    x \leftarrow 1:3; \text{names}(x) \leftarrow c("a", "b", "c") 
    \]

• You can then use the names to access (subset) vector elements:
  
  \[
  x \left[ c("b", "a") \] 
  \]
Why is this useful?

- Because if you know the name (i.e. your label) then you don’t have to remember which element of a vector the data you are after was stored in. Consider this *fictional* example:

```r
> grades <- c(alice=80, barry=99, chandra=60, chris=100)
> grades["barry"]
barry
  99
> which.max(grades)
chris
  4
> sort(grades)
chandra alice barry chris
  60   80   99  100
```
What would happen?

1. ```r
   x <- 1:3; names(x) <- c("a", "b", "c", "d")
```

2. ```r
   x <- 1:3; names(x) <- 3:1; x[3]
```

3. ```r
   x["3"]
```
R has many data structures

These include:

- vector
- data frame
- list
- matrix
- factors
data.frame

• **data.frame** is the *de facto* data structure for most **tabular data** and what we use for statistics and plotting with **ggplot2** - more on this later!

• Arguably the most important R data structure

• Data frames can have additional attributes such as **rownames()** and **colnames()**, which can be useful for annotating data, with things like **subject_id** or **sample_id**
data.frame continued…

• Created with the function `data.frame()`

```r
dat <- data.frame(id = letters[1:10], x = 1:10, y = 11:20)
```

• Or more commonly when reading delimited files (i.e. importing data) with the functions `read.csv()`, `read.table()`, `read_xlsx()` etc…

```r
dep <- read.csv2("http://bio3d.uib.no/data/pdb_deposition2.csv")
```

• R Studio can do this for you via:
  ```
  File > Import Dataset > From CSV…
  ```
Useful `data.frame` Functions

- **`head()`** - and **`tail()`** shows first 6 rows and last 6 rows respectively.
- **`dim()`** - returns the dimensions (i.e. number of rows and columns).
- **`nrow()`** and **`ncol()`** returns the number of rows and columns separately.
- **`rownames()`** and **`colnames()`** - shows the names attribute for rows and columns.
- **`str()`** - returns the structure including name, type and preview of data in each column.
Topics Covered:

Calling Functions
Getting help in R
Vectors and vectorization
Workspace and working directory
RStudio projects
Topics Covered:

Calling Functions
Getting help in R
Vectors and vectorization
Workspace and working directory
RStudio projects
Side-note: Use the code editor for \textbf{R} scripts
R scripts

• A simple text file with your R commands (e.g. lecture7.r) that contains your R code for one complete analysis

• **Scientific method**: complete record of your analysis

• **Reproducible**: rerunning your code is easy for you or someone else

• In RStudio, select code and type `<ctrl+enter>` to run the code in the R console

• **Key point**: Save your R script!
Side-note: RStudio shortcuts

- Sends entire file to console
- Re-send the lines of code you last ran to the console (useful after edits)
- Sends current line or selection to console (faster to type: `command/ctrl+enter`)
- Other RStudio shortcuts!
  - Up/Down arrows (recall cmds)
  - Ctrl + 2 (move cursor to console)
  - Ctrl + 1 (move cursor to editor)
**Rscript:** Third way to use R

1. Terminal
2. RStudio
3. Rscript

**From the command line!**

```
> Rscript --vanilla my_analysis.R
```

# or within R: `source("my_analysis.R")`
R workspaces

• When you close RStudio, **SAVE YOUR .R SCRIPT**

• You can also save data and variables in an R workspace, but this is generally not recommended

• Exception: working with an enormous dataset

• Better to start with a clean, empty workspace so that past analyses don’t interfere with current analyses

• `rm(list = ls())` clears out your workspace

• You should be able to reproduce everything from your R script, so **save your R script, don’t save your workspace!**
R workspaces

- Set Tools > Global Options
RStudio Projects

- We will use a new RStudio **project** for each new class going forward.

  **File > New Project > New Directory > New Project…**

- These projects will help keep us **organized** and divide our work into multiple contexts, each with their own working directory, workspace, history, and source documents.
Learning Resources

- **TryR**. An excellent interactive online R tutorial for beginners. 
  <http://tryr.codeschool.com/>

- **RStudio**. A well designed reference card for RStudio. 
  <https://help.github.com/categories/bootcamp/>

- **DataCamp**. Online tutorials using R in your browser.
  <https://www.datacamp.com/>

  <http://r4ds.had.co.nz/>
Learning Resources

- **TryR**. An excellent interactive online R tutorial for beginners.  
  <http://tryr.codeschool.com/>

- **RStudio**. A well designed reference card for RStudio.  
  <https://help.github.com/categories/bootcamp/>

- **DataCamp**. Online tutorials using R in your browser.  
  <https://www.datacamp.com/>

  <http://r4ds.had.co.nz/>
What is an IDE anyway?

RStudio is an IDE that makes R easier to use by combining a set of tools into a single environment.

What does IDE stand for?

Possible Answers

- Intensive Design Environment
- Integrated Document Environment
- Independent Developer Environment
- Ecosystem

Integrate Development Environment

Submit Answer

<https://www.datacamp.com/>
What is an IDE anyway?

Exercise Completed

Nice job! Move onto the next video to start learning more about the RStudio IDE!

Possible Answers

PRESS ENTER TO CONTINUE

Become a power user!

Submit Answer Ctrl + Shift + Enter

See all keyboard shortcuts

Environment

R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and 'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
## Foundations of Bioinformatics (BGGN-213)

### My Assignments

<table>
<thead>
<tr>
<th>Name</th>
<th>Assigned At</th>
<th>Due By</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditionals and Control Flow</td>
<td>Oct 2, 2017</td>
<td>Nov 2, 2017</td>
<td>In progress</td>
</tr>
<tr>
<td>Introduction to R</td>
<td>Oct 2, 2017</td>
<td>Oct 26, 2017</td>
<td>In progress</td>
</tr>
<tr>
<td>Working with the RStudio IDE (Part 1)</td>
<td>Oct 2, 2017</td>
<td>Oct 26, 2017</td>
<td>In progress</td>
</tr>
</tbody>
</table>
Key Points

• R’s basic data types are **logical**, **character**, **numeric**, integer and complex.

• R’s basic data structures include **vectors**, lists, **data frames**, matrices and factors.

• Objects may have attributes, such as **name**, **dimension**, and **class**.

• **DataCamp**, StackOverflow and **help()** are your friends.
Final Knowledge Check!

- What is R and why should we use it?
- Familiarity with R’s basic syntax.
- Familiarity with major R data structures namely vectors and data.frames (with more on lists and matrices next day).
- Understand the basics of using functions (arguments, vectorization and re-cycling).
- Be able to use R to read and parse comma-separated (.csv) formatted files ready for subsequent analysis.
- Appreciate how you can use R scripts to aid with reproducibility.
Link: Muddy point assessment
http://swcarpentry.github.io/r-novice-inflammation/

Sections: 1, 11 & 12 only!
Help from within R

• Getting help for a function
  ```
  > help("log")
  > ?log
  ```

• Searching across packages
  ```
  > help.search("logarithm")
  ```

• Finding all functions of a particular type
  ```
  > apropos("log")
  [7] "SSlogis" "as.data.frame.logical" "as.logical"
      "as.logical.factor" "dlogis" "is.logical"
  [13] "log" "log10" "log1p" "log2" "logLik" "logb"
  [19] "logical" "loglin" "plogis" "print.logLik" "qlogis"
      "rlogis"
Logarithms and Exponentials

**Description** What the function does in general terms

log computes logarithms, by default natural logarithms. log10 computes common (i.e., base 10) logarithms, and log2 computes binary (i.e., base 2) logarithms. The general form log(x, base) computes logarithms with base base.

log1p(x) computes log(1+x) accurately also for |x| << 1 (and less accurately when x is approximately -1).

exp computes the exponential function.

expm1(x) computes exp(x) - 1 accurately also for |x| << 1.

**Usage** How to use the function

log(x, base = exp(1))
log10(x)
log2(x)
log1p(x)
exp(x)
expm1(x)

**Arguments** What does the function need

x a numeric or complex vector.
base a positive or complex number: the base with respect to which logarithms are computed. Defaults to e=exp(1).

Details

All except logb are generic functions: methods can be defined for them individually or via the base group generic.

log10 and log2 are only convenience wrappers, but logs to bases 10 and 2 (whether computed via log or the wrappers) will be computed more efficiently and accurately where supported by the OS. Methods can be set for them individually (and otherwise methods for log will be used).

logb is a wrapper for log for compatibility with S: if S3 or S4 methods are set for log they will be dispatched. Do not set S4 methods on logb itself.

All except log are primitive functions.

**Value** What does the function return

A vector of the same length as x containing the transformed values. log(0) gives -Inf, and log(x) for negative values of x is NaN. exp(-Inf) is 0.

For complex inputs to the log functions, the value is a complex number with imaginary part in the range [pi, -pi] (which end of the range is used might be platform-specific).

**S4 methods**

exp, expm1, log, log10, log2 and log1p are S4 generic and are members of the Math group generic.

Note that this means that the S4 generic for log has a signature with only one argument, x, but that base can be passed to methods (but will not be used for method selection). On the other hand, if you only set a method for the Math group generic then base argument of log will be ignored for your class.

**Source**

log1p and expm1 may be taken from the operating system, but if not available there are based on the Fortran subroutine dinel by W. Fullerton of Los Alamos Scientific Laboratory (see http://www.netlib.org/slatec/dinel) and (for small x) a single Newton step for the solution of log1p(y) = x respectively.

**References**


**See Also** Discover other related functions

Trig, sqrt, Arithmetic

**Examples** Sample code showing how it works

```r
log(exp(3))
log10(1e7) # = 7
x <- 10^-1(-1+2i)
cbind(x, log(1+x), log1p(x), exp(x)-1, expm1(x))
```

[Package base version 3.0.1 index]
Optional Exercise

Use R to do the following. Create a new script to save your work and code up the following four equations:

1. $1 + 2(3 + 4)$
2. $\ln(4^3 + 3^{2+1})$
3. $\sqrt{(4 + 3)(2+1)}$
4. $\left(\frac{1+2}{3+4}\right)^2$