

BIMM-143: INTRODUCTION TO BIOINFORMATICS (Lecture 11)

Structural Bioinformatics (Part 1)

https://bioboot.github.io/bimm143_W18/lectures/#11

Dr. Barry Grant

Section 1: Introduction to the RCSB Protein Data Bank (PDB)

The PDB archive is the major repository of information about the 3D structures of large biological molecules, including proteins and nucleic acids. Understanding the shape of these molecules helps to understand how they work. This knowledge can be used to help deduce a structure's role in human health and disease, and in drug development. The structures in the PDB range from tiny proteins and bits of DNA or RNA to complex molecular machines like the ribosome composed of many chains of protein and RNA.

In the first section of this lab we will interact with the main US based PDB website (note there are also sites in Europe and Japan).

Visit: <http://www.rcsb.org/> and answer the following questions

NOTE: The “Analyze” -> “PDB Statistics” on the PDB home page should allow you to determine most of these answers.

Q1: What proportion of PDB entries does X-ray crystallography account for? What proportion of structures are protein?

Q2: Type **HIV** in the search box on the home page and determine how many HIV-1 protease structures are in the current PDB?

Now download the “**PDB File**” for the HIV-1 protease structure with the PDB identifier **1HSG**. On the website you can “*Display*” the contents of this “*PDF format*” file. Alternatively, you can examine the contents of your downloaded file in a suitable text editor.



NOTE: On Windows you can use **NotePad** whilst on Mac you can try using the **Terminal** application. For the later open Terminal and try the following command:

```
> more ~/Downloads/1hsg.pdb          ## (use 'q' to quit)
```

NOTE: You can type **1HSG** in the PDB search box to jump to its entry and then click “**Download Files**” to the right of the top display. Selecting “**Display Files**” will allow you to view the PDB file directly in your browser window.

When viewing the file stop when you come the lines beginning with the word “**ATOM**”. We will discuss this ubiquitous PDB file format when you have got this far.

Section 2: Visualizing the HIV-1 protease structure

The HIV-1 protease [1] is an enzyme that is vital for the replication of HIV. It cleaves newly formed polypeptide chains at appropriate locations so that they form functional proteins. Hence, drugs that target this protein could be vital for suppressing viral replication. A handful of drugs - called *HIV-1 protease inhibitors* (saquinavir, zidovudine, zalcitabine, didanosine, zalcitabine, etc.) [2] - are currently commercially available that inhibit the function of this protein, by binding in the catalytic site that typically binds the polypeptide.

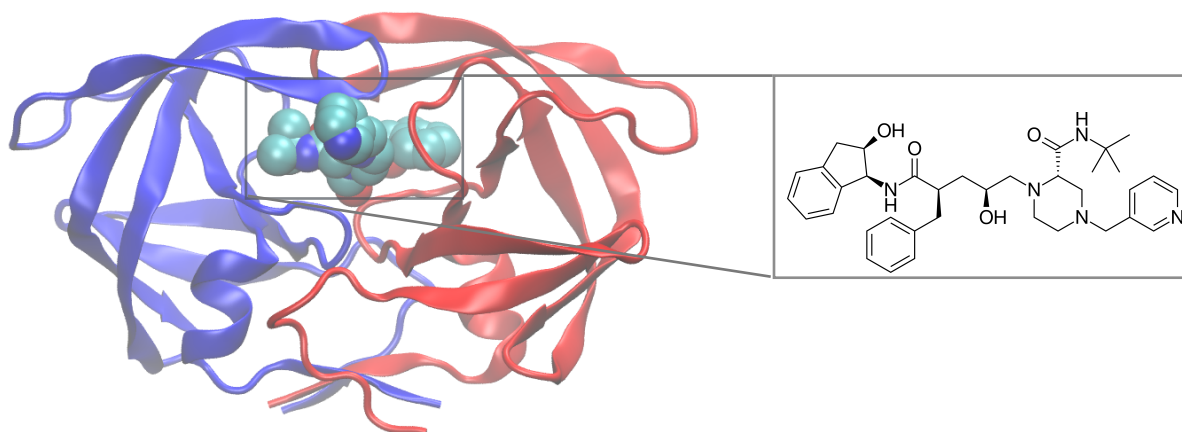


Figure 1. HIV-1 protease structure in complex with the small molecule indinavir.

In this section we will use the 2Å resolution X-ray crystal structure of HIV-1 protease with a bound drug molecule [indinavir](#) (PDB ID: 1HSG) [3]. We will use the **VMD molecular viewer** to visually inspect the protein, the binding site and the drug molecule. After exploring features of the complex we will move on to computationally dock a couple of drug molecules into the binding site of HIV-1 protease to see how well computational docking can reproduce the crystallographically observed binding pose. If time permits, we will also explore the conformational dynamics and flexibility of the protein - important for its function and for considering during drug design.

NOTE: If you have not already done so please download and install VMD from:
<http://www.ks.uiuc.edu/Development/Download/download.cgi> .

Open VMD and load **1hsg.pdb** by using the **VMD Main** window and going to "**File**" -> "**New Molecule**" and then from the new window that appears click "**Browse**" and select your downloaded **1hsg.pdb** file. Then click "**Load**".

You should now see the protein structure displayed as lines and water molecules as little red dots. Use the mouse to zoom and rotate. Once you have the hang of rotation we will start exploring different "*Graphical Representations*".

VMD can display molecules in various ways by choosing different options in the *Graphical Representations* window shown in **Figure 2**. You can access this window by clicking **Graphics > Representations** from the small **VMD Main** window.

NOTE. Each representation is defined by three main parameters: (1) the *drawing method*, (2) the *selected atoms* to be included in the representation, and (3) the *coloring method* (see Figure 2 labels 1-3)

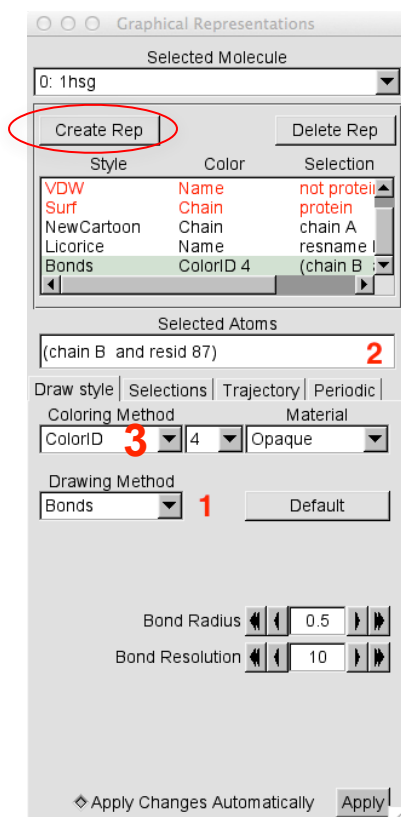
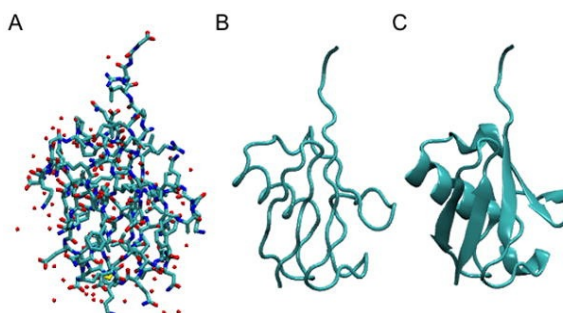


Figure 2. The VMD graphical Representation window. Note that (1) the *drawing method* defines which graphical representation is used and (2) the selection determines which part of the molecule is drawn, and (3) defines the color it is displayed with. You are encouraged to explore different drawing styles (*Drawing Methods* - labeled 1) including *Licorice*, *Tube* and *NewCartoon* (see below for examples A-C).



Also try different selections by entering text in the (*Selected Atoms* box - labeled 2). Some examples to try include:

```
chain A and backbone
resname ASP
within 5 of resname MK1
```

Using Atom Selections

Now type “protein” in the *Selected Atoms* text box (labeled 2 in Figure 2) and show the protein using the **Cartoon** representation and color by **chain** (see label 3 in Figure 2.)

Lets add a new representation by clicking the “**Create Rep**” (circled in Figure 2) and using the selection text “**not protein and not water**”

Add more representations (by clicking the “**Create Rep**” button) and hiding (by double clicking) or deleting previous ones (with the “**Delete Rep**” button) to explore different representations for both the ligand and the protein.

NOTE: you can use the residue name of the ligand “resname MK1” to select just the ligand.

Water molecules have the residue name HOH. Select and display all water molecules as red spheres. If you think the spheres are too big, how would you reduce their size?

Q3: Water molecules normally have 3 atoms. Why do we see just one atom per water molecule in this structure?

Q4: There is a conserved water molecule in the binding site. Can you identify this water molecule? What residue number does this water molecule have (see note below)?

NOTE: From the **VMD Main** window click **Mouse > Label > Atoms** and then click on the water in question to display its residue number. A short cut is to press the #1 key when your mouse is active in the OpenGL window.

Now you should be able to produce an image similar or even superior to **Figure 1** and save it to an image file on disk with **VMD Main** window, **File > Render > Start Rendering**.

NOTE: You can choose different rendering engines including Tachyon (internal), which is commonly used for publication quality images.

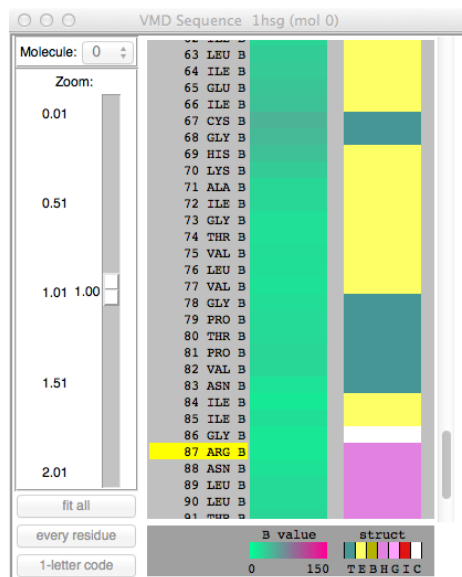
Optional: Generate and save a figure clearly showing the two distinct chains of HIV-protease along with the ligand. You might also consider showing the catalytic residues ASP 25 in each chain (we recommend Licorice for these side-chains). Email this figure to bjgrant@ucsd.edu for grading.

Discussion Topic: Can you think of a way in which indinavir, or even larger ligands and substrates, could enter the binding site?

Sequence Viewer Extension

When dealing with a protein for the first time, it is very useful to be able to find and display different amino acids quickly. The sequence viewer extension allows viewing of the protein sequence, as well as to easily pick and display one or more residues of interest.

To launch the Sequence Viewer click **VMD Main** window, **Extensions > Analysis > Sequence Viewer**. The different color scales beside the sequence correspond to the B-factor and Secondary structure type (the major ones being Extended (beta) in yellow and Helix in purple).



Q5: List the residue numbers and primary sequence in one-letter code of helix positions in chain B.

Secondary Structure codes used by STRIDE.	
Letter Code	Secondary Structure
T	Turn
E	Extended conformation (β -sheets)
B	Isolated bridge
H	Alpha helix
G	3–10 helix
I	Pi helix
C	Coil

Q6: As you have hopefully observed HIV protease is a homodimer (i.e. it is composed of two identical chains). With the aid of the graphic display and the sequence viewer extension can you identify secondary structure elements that are likely to only form in the dimer rather than the monomer?

Section3: Introduction to Bio3D in R

Bio3D¹ is an R package containing utilities for the analysis of biomolecular structure, sequence and trajectory data ^{2,3}. Features include the ability to read and write biomolecular structure, sequence and dynamic trajectory data, perform atom selection, re-orientation, superposition, rigid core identification, clustering, distance matrix analysis, conservation analysis, normal mode analysis and principal component analysis. Bio3D takes advantage of the extensive graphical and statistical capabilities of the [R environment](#) and thus represents a useful framework for exploratory analysis of structural data.

3.1 Getting started

Start R (type R at the command prompt or, double click on the **R-studio** icon) and load the Bio3D package by typing `library(bio3d)` at the R console prompt:

```
library(bio3d)
```

Side-Note: If you see an error message reported then you will first need to install the package with the command: `install.packages("bio3d")`. This is only required once whereas the `library(bio3d)` command is required at the start of every new R session where you want to use Bio3D.

Then use the command `lbio3d()` or `help(package=bio3d)` to list the functions within the package and `help(FunctionName)` to obtain more information about an individual function.

¹ The latest version of the Bio3D package, full documentation and further vignettes (including detailed installation instructions) can be obtained from the main Bio3D website: thegrantlab.org/bio3d/.

² Grant, B.J. et al. (2006) *Bioinformatics* 22:2695--2696.

³ Skjaerven, L. et al. (2014) *BMC Bioinformatics* 15:399.

```
# List of bio3d functions with brief description
```

```
help(package=bio3d)
```

```
# Detailed help on a particular function, e.g. 'pca.xyz'
```

```
help(pca.xyz)
```

To search the help system for documentation matching a particular word or topic use the command `help.search("topic")`. For example, `help.search("pdb")`

```
help.search("pdb")
```

Side-note: You can find online documentation at <https://www.rdocumentation.org/packages/bio3d/versions/2.3-3> and at the official [Bio3D web-site](#).

3.2 Bio3D functions and their typical usage

The Bio3D package consists of input/output functions, conversion and manipulation functions, analysis functions, and graphics functions all of which are fully documented both [online](#) and within the R help system introduced above.

To better understand how a particular function operates it is often helpful to view and execute an example. Every function within the Bio3D package is documented with example code that you can view by issuing the `help()` command.

Running the command `example(function)` will directly execute the example for a given function. In addition, a number of longer worked examples are available as [Tutorials](#) on the Bio3D website.

```
example(plot.bio3d)
```

3.3 Working with individual PDB files

Protein Data Bank files (or PDB files) are the most common format for the distribution and storage of high-resolution biomolecular coordinate data. The Bio3D package contains functions for the reading (e.g. `read.pdb()`, `get.pdb()`), writing (`write.pdb()`) and manipulation (e.g. `trim.pdb()`, `atom.select()`, `pdbsplit()`) of PDB files. Indeed numerous Bio3D analysis functions are intended to operate on PDB file derived data (e.g. `blast.pdb()`, `pdbaln()`, `hmmer()`, `pca()`, `nma()`, `dssp()`, etc.)

At their most basic, PDB coordinate files contain a list of all the atoms of one or more molecular structures. Each atom position is defined by its x, y, z coordinates in a conventional orthogonal coordinate system. Additional data, including listings of observed secondary structure elements, are also commonly (but not always) detailed in PDB files.

Reading PDB file data into R

To read a single PDB file with Bio3D we can use the `read.pdb()` function. The minimal input required for this function is a specification of the file to be read. This can be either the file name of a local file on disc, or the RCSB PDB identifier of a file to read directly from the on-line PDB repository. For example to read and inspect the on-line file with PDB ID 4q21:

```
pdb <- read.pdb("4q21")

## Note: Accessing on-line PDB file
```

To get a quick summary of the contents of the `pdb` object you just created you can issue the command `print(pdb)` or simply type `pdb` (which is equivalent in this case):

```
pdb

##
## Call: read.pdb(file = "4q21")
##
## Total Models#: 1
## Total Atoms#: 1447, XYZs#: 4341 Chains#: 1 (values: A)
##
## Protein Atoms#: 1340 (residues/Calpha atoms#: 168)
## Nucleic acid Atoms#: 0 (residues/phosphate atoms#: 0)
##
## Non-protein/nucleic Atoms#: 107 (residues: 80)
## Non-protein/nucleic resid values: [ GDP (1), HOH (78), MG (1) ]
##
## Protein sequence:
## MTEYKLVVVGAGGVGKSALTIQLIQNHFVDEYDPTIEDSYRKQVVIDGETCLLDILDITAG
## QEEYSAMRDQYMRTGEGFLCVFAINNTKSFEDIHQYREQIKRVKDSDDVPMVLVGNKCDL
## AARTVESRQAQDLARSYGIPYIETSAKTRQGVEDAFYTLVREIRQHKL
##
## + attr: atom, xyz, seqres, helix, sheet,
## calpha, remark, call
```

Q7. How many amino acid residues are there in this `pdb` object and what are the three non-protein residues?

Note that the attributes (`+ attr:`) of this object are listed on the last couple of lines. To find the attributes of any such object you can use:

```
attributes(pdb)

## $names
## [1] "atom" "xyz" "seqres" "helix" "sheet" "calpha" "remark" "call"
##
```

```
## $class
## [1] "pdb" "sse"
```

To access these individual attributes we use the dollar-attribute name convention that is common with R list objects. For example, to access the atom attribute or component use `pdb$atom`:

```
head(pdb$atom)
```

```
##   type eleno elety alt resid chain resno insert      x      y      z o
## 1 ATOM     1     N <NA>  MET     A     1  <NA> 64.080 50.529 32.509 1
## 2 ATOM     2     CA <NA>  MET     A     1  <NA> 64.044 51.615 33.423 1
## 3 ATOM     3     C  <NA>  MET     A     1  <NA> 63.722 52.849 32.671 1
## 4 ATOM     4     O <NA>  MET     A     1  <NA> 64.359 53.119 31.662 1
## 5 ATOM     5     CB <NA>  MET     A     1  <NA> 65.373 51.805 34.158 1
## 6 ATOM     6     CG <NA>  MET     A     1  <NA> 65.122 52.780 35.269 1
##. <... cut for brevity ...>
```

```
# Print $atom data for the first two atoms
```

```
pdb$atom[1:2, ]
```

```
##   type eleno elety alt resid chain resno insert      x      y      z o
## 1 ATOM     1     N <NA>  MET     A     1  <NA> 64.080 50.529 32.509 1
## 2 ATOM     2     CA <NA>  MET     A     1  <NA> 64.044 51.615 33.423 1
##. <... cut for brevity ...>
```

```
# Print a subset of $atom data for the first two atoms
```

```
pdb$atom[1:2, c("eleno", "elety", "x", "y", "z")]
```

```
##   eleno elety      x      y      z
## 1     1     N 64.080 50.529 32.509
## 2     2     CA 64.044 51.615 33.423
```

```
# Note that individual $atom records can also be accessed like this
```

```
pdb$atom$elety[1:2]
```

```
## [1] "N" "CA"
```

```
# Which allows us to do the following
```

```
plot.bio3d(pdb$atom$b[pdb$alpha], sse=pdb, typ="l", ylab="B-factor")
```

Q8. What type of R object is `pdb$atom`? **HINT:** You can always use the `str()` function to get a useful summary of any R object.

Note that the main xyz coordinate attribute is a numeric matrix with 3N columns (each atom has three values x, y and z). The number of rows here correspond to the number of models in the PDB file (typically one for X-ray structures and multiple for NMR structures).

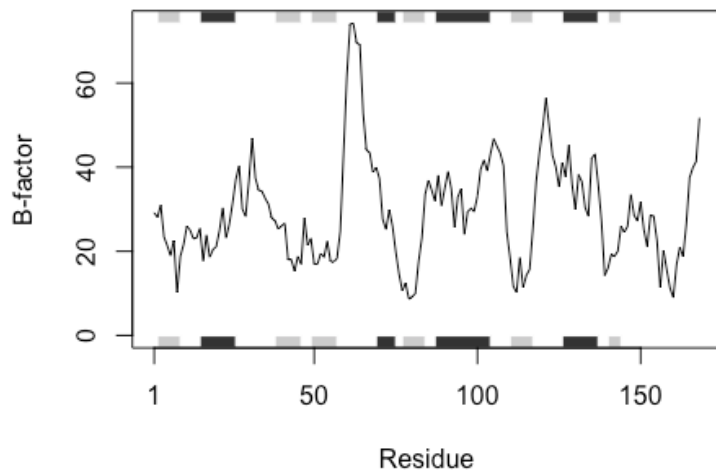


Figure 5: Residue temperature factors for PDB ID 4q21 with secondary structure element (SSE) annotation in marginal regions plotted with function `plot.bio3d()`

Q9. What type of secondary structure is found at the protein regions displaying the highest B-factors?

```
# Print a summary of the coordinate data in $xyz
pdb$xyz

##
##   Total Frames#: 1
##   Total XYZs#:   4341, (Atoms#:  1447)
##
##   [1]  64.08  50.529  32.509  <...>  74.159  76.923  41.999  [4341]
##
## + attr: Matrix DIM = 1 x 4341

# Examine the row and column dimensions
dim(pdb$xyz)

## [1]      1 4341

# Print coordinates for the first two atom
pdb$xyz[ 1, atom2xyz(1:2) ]

## [1] 64.080 50.529 32.509 64.044 51.615 33.423
```

Side-Note: The 'pdb' class. Objects created by the `read.pdb()` function are of class "pdb". This is recognized by other so called generic Bio3D functions (for example `atom.select()`, `nma()`, `print()`, `summary()` etc.). A generic function is a function that examines the class of its first argument, and then decides what type of operation to perform (more specifically it decides which specific method to dispatch to). So for example, the generic `atom.select()` function knows that the input is of class "pdb", rather than for example an AMBER parameter and topology file, and will act accordingly.

A careful reader will also of noted that our "pdb" object created above also has a second class, namely "sse" (see the output of `attributes(pdb)` or `class(pdb)`). This stands for *secondary structure elements* and is recognized by the `plot.bio3d()` function to annotate the positions of major secondary structure elements in the marginal regions of these plots (see Figure 1). This is all part of the R S3 object orientation system. This S3 system os used throughout Bio3D to simplify and facilitate our work with these types of objects.

3.5 Atom selection

The Bio3D `atom.select()` function is arguably one of the most challenging for newcomers to master. It is however central to PDB structure manipulation and analysis. At its most basic, this function operates on PDB structure objects (as created by `read.pdb()`) and returns the numeric indices of a selected atom subset. These indices can then be used to access the `$atom` and `$xyz` attributes of PDB structure related objects.

For example to select the indices for all C-alpha atoms we can use the following command:

```
# Select all C-alpha atoms (return their indices)
ca.inds <- atom.select(pdb, "calpha")
ca.inds

##
## Call:  atom.select.pdb(pdb = pdb, string = "calpha")
##
##      Atom Indices#: 168  ($atom)
##      XYZ   Indices#: 504  ($xyz)
##
## + attr: atom, xyz, call
```

Note that the attributes of the returned `ca.inds` from `atom.select()` include both `atom` and `xyz` components. These are numeric vectors that can be used as indices to access the corresponding atom and xyz components of the input PDB structure object. For example:

```
# Print details of the first few selected atoms
head( pdb$atom[ca.inds$atom, ] )
```

```
##      type eleno elety  alt resid chain resno insert      x      y      z o
## 2  ATOM      2    CA <NA>  MET      A      1  <NA> 64.044 51.615 33.423 1
## 10 ATOM     10    CA <NA>  THR      A      2  <NA> 62.439 54.794 32.359 1
## 17 ATOM     17    CA <NA>  GLU      A      3  <NA> 63.968 58.232 32.801 1
## 26 ATOM     26    CA <NA>  TYR      A      4  <NA> 61.817 61.333 33.161 1
## 38 ATOM     38    CA <NA>  LYS      A      5  <NA> 63.343 64.814 33.163 1
## 47 ATOM     47    CA <NA>  LEU      A      6  <NA> 61.321 67.068 35.557 1
##. <... cut for brevity ...>
```

And selected xyz coordinates

```
head( pdb$xyz[, ca.ind$xyz] )
```

```
## [1] 64.044 51.615 33.423 62.439 54.794 32.359
```

In addition to the common selection strings (such as 'calpha' 'cbeta' 'backbone' 'protein' 'notprotein' 'ligand' 'water' 'notwater' 'h' and 'noh') various individual atom properties can be used for selection.

Select chain A

```
a.ind$ <- atom.select(pdb, chain="A")
```

Select C-alphas of chain A

```
ca.ind$ <- atom.select(pdb, "calpha", chain="A")
```

We can combine multiple selection criteria to return their intersection

```
cab.ind$ <- atom.select(pdb, elety=c("CA","CB"), chain="A",
resno=10:20)
```

3.6 Write a PDB object

Use the command `write.pdb()` to output a structure object to a PDB formatted file on your local hard drive. Below we use function `atom.select()` to select only the backbone atoms, and `trim.pdb()` to create a new PDB object based on our selection of backbone atoms. Finally we use the function `write.pdb()` to write the newly generated PDB file containing only the backbone atoms:

Output a backbone only PDB file to disc

```
b.ind$ <- atom.select(pdb, "back")
backpdb <- trim.pdb(pdb, b.ind$)
write.pdb(backpdb, file="4q21_back.pdb")
```

Side-note: The selection statement can directly be provided into function `trim.pdb()`. Alternatively, function `atom.select()` can also return the resulting trimmed `pdb` object using the optional argument `value=TRUE`. See examples below:

```
# Selection statements can be passed directly to trim.pdb()
backpdb <- trim.pdb(pdb, "backbone")

# The 'value=TRUE' option of atom.select() will result in a PDB object
being returned
backpdb <- atom.select(pdb, "backbone", value=TRUE)
```

Function `write.pdb()` contains further arguments to change the specific data in the PDB structure object. For example, using argument `resno` the residue numbers in the PDB object will be changed according to the input values, e.g. for the purpose of renumbering a PDB object (see also `clean.pdb()` and `convert.pdb()` functions):

```
# Renumber all residues
write.pdb(backpdb, resno=backpdb$atom$resno+10)

# Assign chain B to all residues
write.pdb(backpdb, chain="B")
```

You may wish to open your output file in **VMD**, or another molecular viewer, to ensure it meets your expectations.

Q10. How would you output a PDB file with a file name of your choosing? **HINT:** You can always use look at the help page of any `bio3d` function with the `?write.pdb` command.

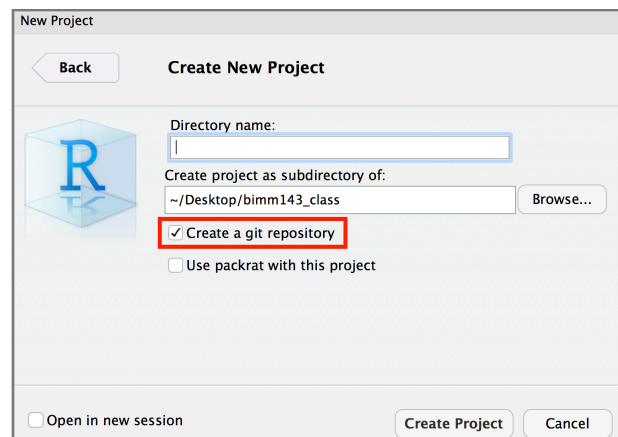
Section 4: Adding Your Version Controlled Project Work to GitHub

Open a new R Studio Project in a sensible working directory and be sure to select the “**create git repository**” option if you have it available. If you don’t have this tick-box option available please let Barry know!

Setup a GitHub R markdown file:

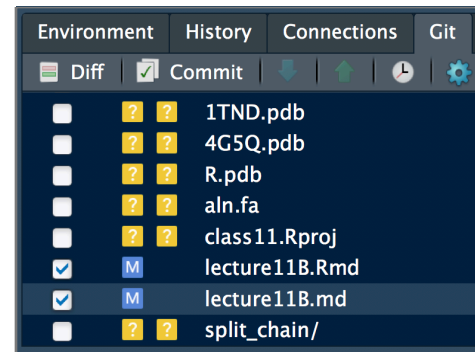
Next, open a new R Markdown document (**File > New File > R Markdown...**). Chose “**From Template**” and select “**GitHub Document**”.

Use this new document to store all your code, output and text notes for subsequent sections. However, be sure to retain the “*setup*” code block and “*output: github_document*” line in the ymal header section.



Be sure to save your file regularly - it will have the file extension **.Rmd**. Once you have some content “**Kinit**” your document to produce a GitHub format markdown file (with file extension **.md**) that we can upload to GitHub using the Git panel of RStudio that we covered last day:

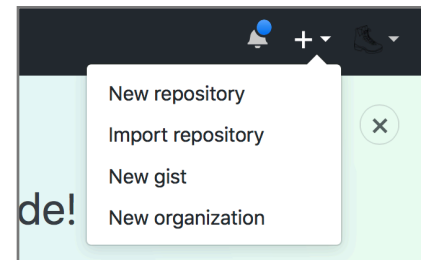
In the Git panel of RStudio click to add both your **.Rmd** and **.md** files. Then click the “Commit” icon; add a commit message and click the “Commit” button.



Create a new GitHub repository:

Visit GitHub (<https://github.com>) and make sure you are logged in with the account you created last day. Create a new repository by clicking the plus **+** icon in the top navigation bar.

Chose a name for your new repo, for example “*class11*” and click **Create Repository**



Add your GitHub repository to your RStudio project

On the resulting GitHub web page copy the text under “...or push an existing repository from the command line”. We will need to add this code to our RStudio project.

Back in RStudio, again click **Tools > Shell ...** . And then paste your copied code in to the shell. My code looked like the below. Yours will be different and depend on your GitHub username and the name of the repository you just setup:

```
git remote add origin git@github.com:bioboot/class11.git
git push -u origin master
```

You can close the shell after this and return to RStudio.

You have now pushed your commit to GitHub, and should be able to see your files in your GitHub account by refreshing the web-page (i.e. reloading). The **Push** and **Pull** arrow buttons in RStudio under the Git icon will now also work.

Side-Note: Remember, after each **Commit**, you have to **Push** to GitHub, this doesn't happen automatically!

Section 5: Working with multiple PDB files

The Bio3D package was designed to specifically facilitate the analysis of multiple structures from both experiment and simulation.

The challenge of working with these structures is that they are usually different in their composition (i.e. contain differing number of atoms, sequences, chains, ligands, structures, conformations etc. even for the same protein as we will see below) and it is these differences that are frequently of most interest.

For this reason Bio3D contains extensive utilities to enable the reading, writing, manipulation and analysis of such heterogenous structure sets. This topic is detailed extensively in the separate **Principal Component Analysis** vignette and **Ensemble Normal Mode Analysis** vignette available from <http://thegrantlab.org/bio3d/tutorials>.

5.1 Installing the stand-alone muscle alignment program

If you are not on one of the classroom computers you will need to download the appropriate version of the **muscle** multiple alignment program from: <https://www.drive5.com/muscle/downloads.htm>

Side-Note: For Mac you will most likely want the Intel i86 64 bit version unless you are on a particularly old laptop. Ask Barry if you have questions about this step

After downloading MUSCLE, it should be unzipped and renamed to just “**muscle**”. If you are on Mac you can move this file to a directory such as “**/usr/local/bin/**”

The easiest way to do this on Mac or Linux is to use the **Terminal/Shell** in your RStudio. At the shell prompt issue the following commands:

```
tar -xvf ~/Downloads/muscle3.8.31_i86darwin32.tar
mv muscle3.8.31_i86darwin32 /usr/local/bin/muscle
```

If you now type `muscle` in your terminal you should see the help splash screen for the MUSCLE program. Instead, if you see the response “`bash: muscle: command not found`” something has not worked and you should ask Barry for help!

5.2 Aligning multiple structures

Before delving into more advanced analysis (detailed in the next section and additional vignettes) lets examine how we can read multiple PDB structures from the RCSB PDB for a particular protein and perform some basic analysis:

```
# Download some example PDB files
ids <- c("1TND_B", "1AGR_A", "1TAG_A", "1GG2_A", "1KJY_A", "4G5Q_A")
files <- get.pdb(ids, split = TRUE)
```

The `get.pdb()` function will download the requested files. Argument `split = TRUE` requests further that we want to extract particular chains, i.e. those specified by the `_A` suffix of each PDB ID in the example above. Note that these `ids` could come from the results of a `blast.pdb()` search as described in other vignettes.

The requested chains are then aligned and their structural data stored in a new object `pdb`s that can be used for further analysis. The `pdbaln()` function includes the ability to superimpose, or fit, all structures onto each other using the argument `fit = TRUE`.

Side-Note: You can also provide a vector of PDB IDs, or a list of **pdb** objects as input to `pdbaln()`. Here we use a vector of file names (output from `get.pdb()`).

```
# Extract and align the chains we are interested in
pdb<= pdbaln(files, fit = TRUE)
```

```
# Print to screen a summary of the 'pdb' object
pdb<=
```

Side-Note: If you are on **Windows** and the call to `pdbaln()` yielded an ERROR message it is likely that you need to provide an extra option to the `pdbaln()` function call to tell R where muscle lives on your computer: For example:

```
pdb<= pdbaln(files, fit = TRUE, exefile="c:/Downloads/muscle.exe")
```

Or if that does not work you can use the online EBI muscle server:

```
pdb<= pdbaln(files, fit=TRUE, web.args=list(email="your@email.com"))
```

Here we use a vector of file names (output from `get.pdb()`).

Q11: What effect does setting the `fit=TRUE` option have in the related `rmsd()` function? What does RMSD measure and what would the results indicate if you set `fit=FALSE` or removed this option? **HINT:** Bio3D functions have various default options that will be used if the option is not explicitly specified by the user, see `help(rmsd)` for an example and note that the input options with an equals sign (e.g. `fit=FALSE`) have default values.

Here the returned object is of class `pdb`s. Note that it contains a `xyz` numeric matrix of aligned C-alpha coordinates, a `ali` matrix of aligned residues, and a `resno` matrix of aligned residue

numbers (see the list of associated attributes (+ attr)). These attributes can be accessed using the common \$ syntax in R. E.g. use `pdb$ali` to access the alignment. To access the first few rows of the alignment matrix we use standard subsetting syntax for matrices in R:

```
# Access the first 5 rows, and 8 columns
```

```
pdb$ali[1:5, 1:8]
```

```
##           [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## ./split_chain/1TND_B.pdb "-"  "-"  "-"  "-"  "-"  "-"  "-"
## ./split_chain/1AGR_A.pdb "L"   "S"   "A"   "E"   "D"   "K"   "A"   "A"
## ./split_chain/1TAG_A.pdb "-"  "-"  "-"  "-"  "-"  "-"  "-"  "-"
## ./split_chain/1GG2_A.pdb "L"   "S"   "A"   "E"   "D"   "K"   "A"   "A"
## ./split_chain/1KJY_A.pdb "-"  "-"  "-"  "-"  "-"  "-"  "-"  "-"
```

```
# Associated residues numbers
```

```
pdb$resno[1:5, 1:8]
```

```
##           [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## ./split_chain/1TND_B.pdb  NA   NA   NA   NA   NA   NA   NA   NA
## ./split_chain/1AGR_A.pdb   5    6    7    8    9   10   11   12
## ./split_chain/1TAG_A.pdb  NA   NA   NA   NA   NA   NA   NA   NA
## ./split_chain/1GG2_A.pdb   5    6    7    8    9   10   11   12
## ./split_chain/1KJY_A.pdb  NA   NA   NA   NA   NA   NA   NA   NA
```

Gap regions are indicated with a - (or NA for `pdb$resno`). You can inspect your alignment for gaps using function `gap.inspect()`. This function provides various indices to obtain an overview of gaps in the alignment object such as indices for non-gap columns (`gaps$f.inds`), gap containing columns (`gaps$t.inds`), gaps per row (`gaps$row`), gaps per column (`gaps$col`). These indices can be used to access the alignment matrix, e.g.:

```
# Inspect alignment for gaps
```

```
gaps <- gap.inspect(pdb$ali)
```

```
# Indices of non-gap columns
```

```
head(gaps$f.inds)
```

```
# Access non-gap columns
```

```
pdb$ali[, gaps$f.inds]
```

Side-Note: The row names of the alignment matrix (`pdb$ali`) as well as the identifiers component (`pdb$id`) is set to the file name of the associated PDB file. You can convert these identifiers to their PDB codes using the `basename.pdb()` function (e.g. `basename.pdb(pdb$id)`).

5.3 Basic structure analysis

Having the generated `pdb`s object at hand facilitates a range of possibilities for protein structure analysis. This includes sequence identity/similarity, structural deviation, rigid core identification as well as principal component and normal mode analysis. Several Bio3D functions are specifically designed to operate on the `pdb`s object, including functions `seqidentity()`, `rmsd()`, `pca()`, `core.find()`, `nma()` and many others.

Below we calculate the pairwise sequence identity between the structures of the `pdb`s ensemble followed by the root mean square deviation (RMSD):

```
# Calculate sequence identity
seqidentity(pdb)

##           1TND_B 1AGR_A 1TAG_A 1GG2_A 1KJY_A 4G5Q_A
## 1TND_B    1.000  0.693  1.000  0.690  0.696  0.696
## 1AGR_A    0.693  1.000  0.694  0.997  0.994  0.997
## 1TAG_A    1.000  0.694  1.000  0.691  0.697  0.697
## 1GG2_A    0.690  0.997  0.691  1.000  0.991  0.994
## 1KJY_A    0.696  0.994  0.697  0.991  1.000  1.000
## 4G5Q_A    0.696  0.997  0.697  0.994  1.000  1.000

# Calculate RMSD
rmsd(pdb)

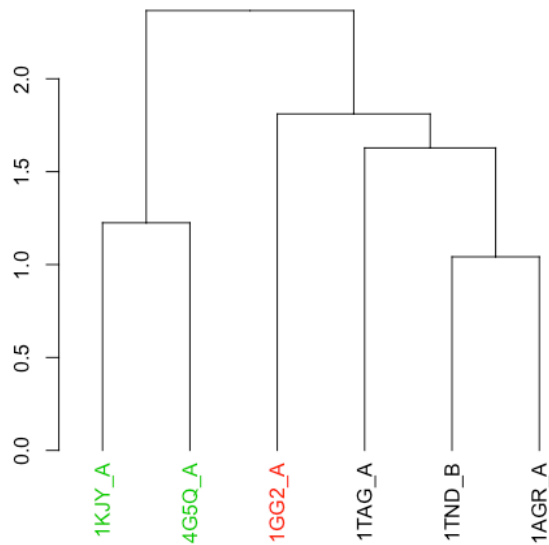
##           1TND_B 1AGR_A 1TAG_A 1GG2_A 1KJY_A 4G5Q_A
## 1TND_B    0.000  1.042  1.281  1.651  2.098  2.367
## 1AGR_A    1.042  0.000  1.628  1.811  1.949  2.244
## 1TAG_A    1.281  1.628  0.000  1.730  1.840  1.885
## 1GG2_A    1.651  1.811  1.730  0.000  1.901  2.032
## 1KJY_A    2.098  1.949  1.840  1.901  0.000  1.225
## 4G5Q_A    2.367  2.244  1.885  2.032  1.225  0.000
```

These pairwise similarity measures facilitate the identification of groups of structures sharing a similar conformation (in case of RMSD) through clustering analysis:

```
# Calculate RMSD
rd <- rmsd(pdb)

# Clustering
hc <- hclust(as.dist(rd))
grps <- cutree(hc, k=3)
```

```
# Plot results as dendrogram
hclustplot(hc, k=3)
```



5.5 Principal component analysis

Function `pca()` provides *principal component analysis* (PCA) of the structure data. PCA is a statistical approach used to transform a data set down to a few important components that describe the directions where there is most variance. In terms of protein structures PCA is used to capture major structural variations within an ensemble of structures.

```
# Perform PCA
pc <- pca(pdbbs)
plot(pc, col=grps)
```

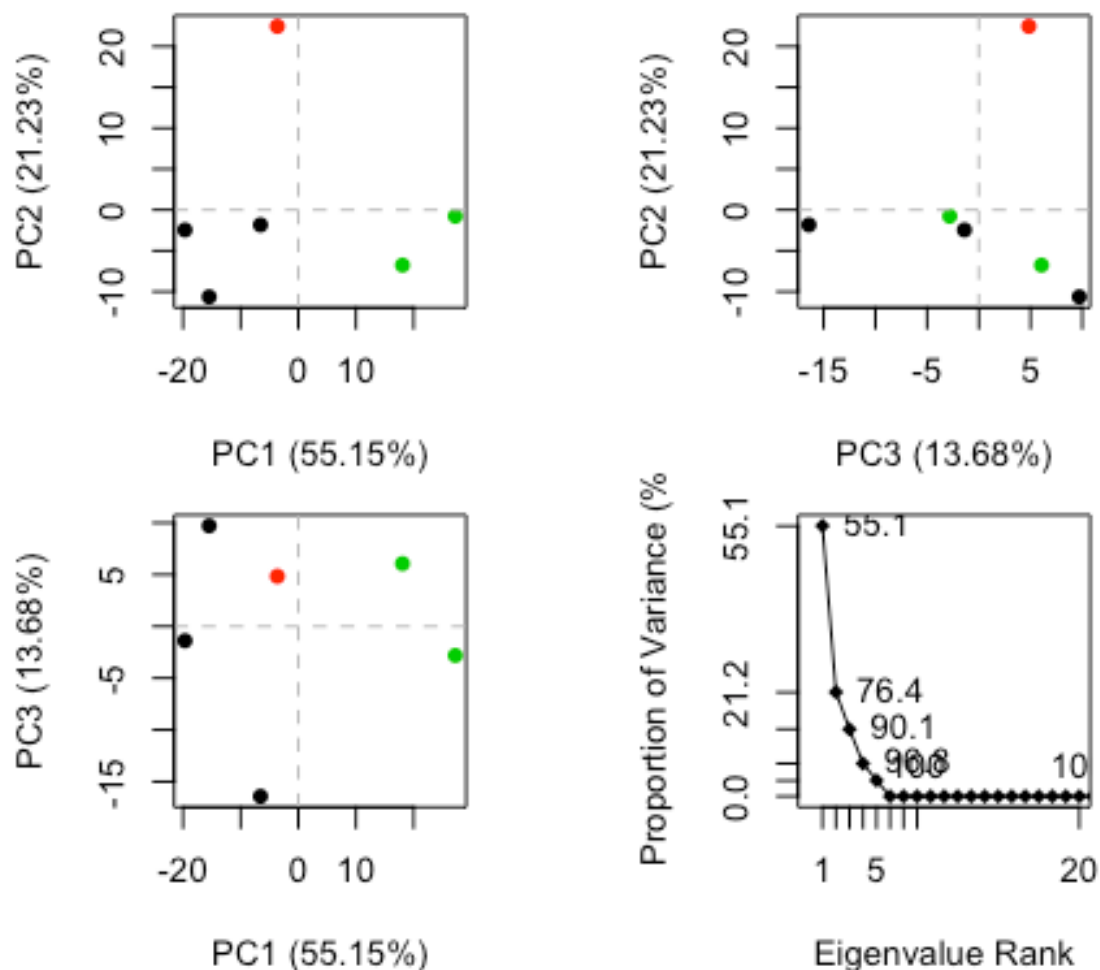


Figure 9: Results of protein structure principal component analysis (PCA) plotted by the function `plot.pca()`

The plot shows a conformer plot -- a low-dimensional representation of the conformational variability within the ensemble of PDB structures. The plot is obtained by projecting the individual structures onto two selected PCs (e.g. PC-1 and PC-2). These projections display the inter-conformer relationship in terms of the conformational differences described by the selected PCs.

5.6 Normal mode analysis

Function `nma()` provides *normal mode analysis* (NMA) on the complete structure ensemble. This facilitates characterising and comparing flexibility profiles of related protein structures.

```
# NMA all structures
modes <- nma(pdb)
plot(modes, pdbs, col=grps, spread=TRUE)
```

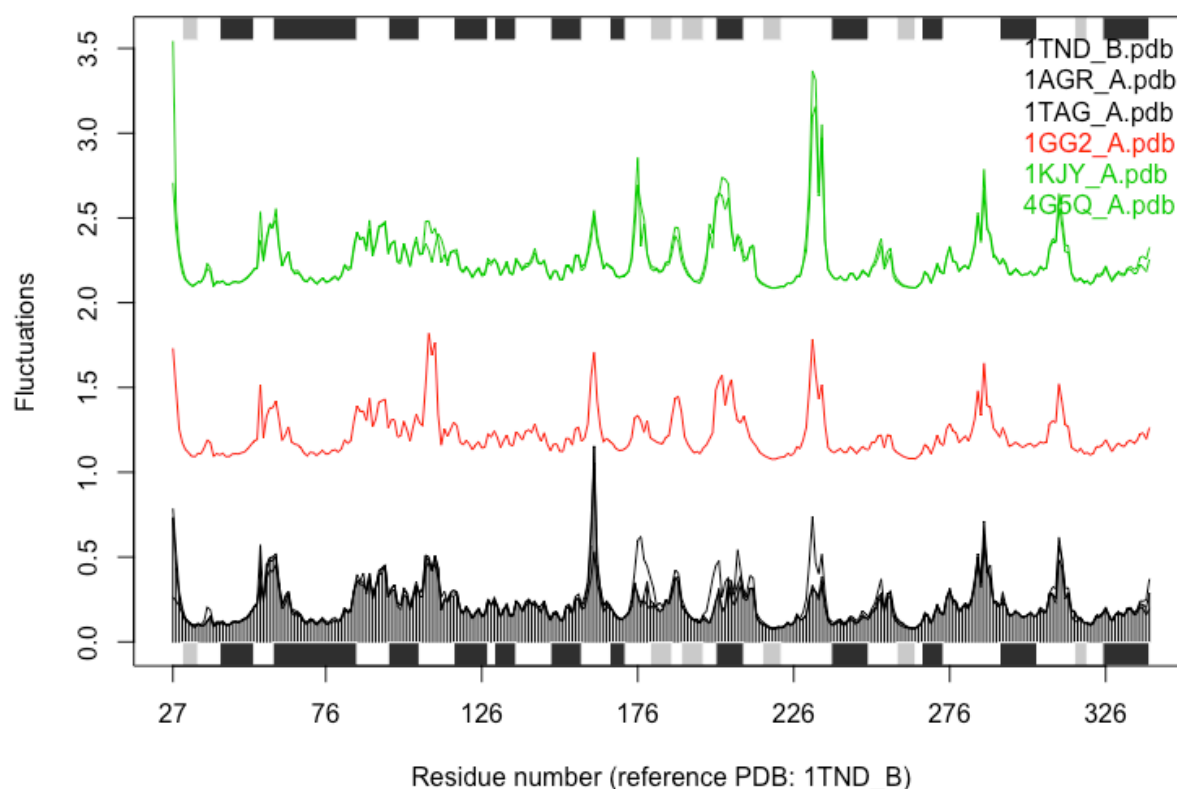


Figure 10: Results of ensemble normal mode analysis (NMA) plotted by function `plot.enma()`

Section 6: Example Application on Adenylate Kinase (Adk)

In this section we perform PCA on the complete collection of Adenylate kinase structures in the protein data-bank (PDB). Starting from only one PDB identifier (PDB ID 1AKE) we show how to search the PDB for related structures using BLAST, fetch and align the structures, and finally calculate the normal modes of each individual structure in order to probe for potential differences in structural flexibility.

6.1 Search and retrieve Adenylate kinase structures

Below we perform a blast search of the PDB database to identify related structures to our query Adenylate kinase sequence. In this particular example we use function `get.seq()` to fetch the query sequence for chain A of the PDB ID 1AKE and use this as input to `hmmerr()`. Note that `get.seq()` would also allow the corresponding UniProt identifier.

```

aa <- get.seq("lake_A")

## Fetching... Please wait. Done.

# Blast or hmmer search
b <- blast.pdb(aa)

## Searching ... please wait (updates every 5 seconds) RID =
ZM7GP50C014
## .
## Reporting 209 hits

```

Function `plot.blast()` facilitates the visualization and filtering of the Blast results. It will attempt to set a seed position to the point of largest drop-off in normalized scores (i.e. the biggest jump in E-values). In this particular case we specify a cutoff (after initial plotting) of 225 to include only the relevant *E.coli* structures:

```

# Plot a summary of search results
hits <- plot(b)

## * Possible cutoff values:      198 -3
##           Yielding Nhits:      39 209
##
## * Chosen cutoff value of:      198
##           Yielding Nhits:      39

```

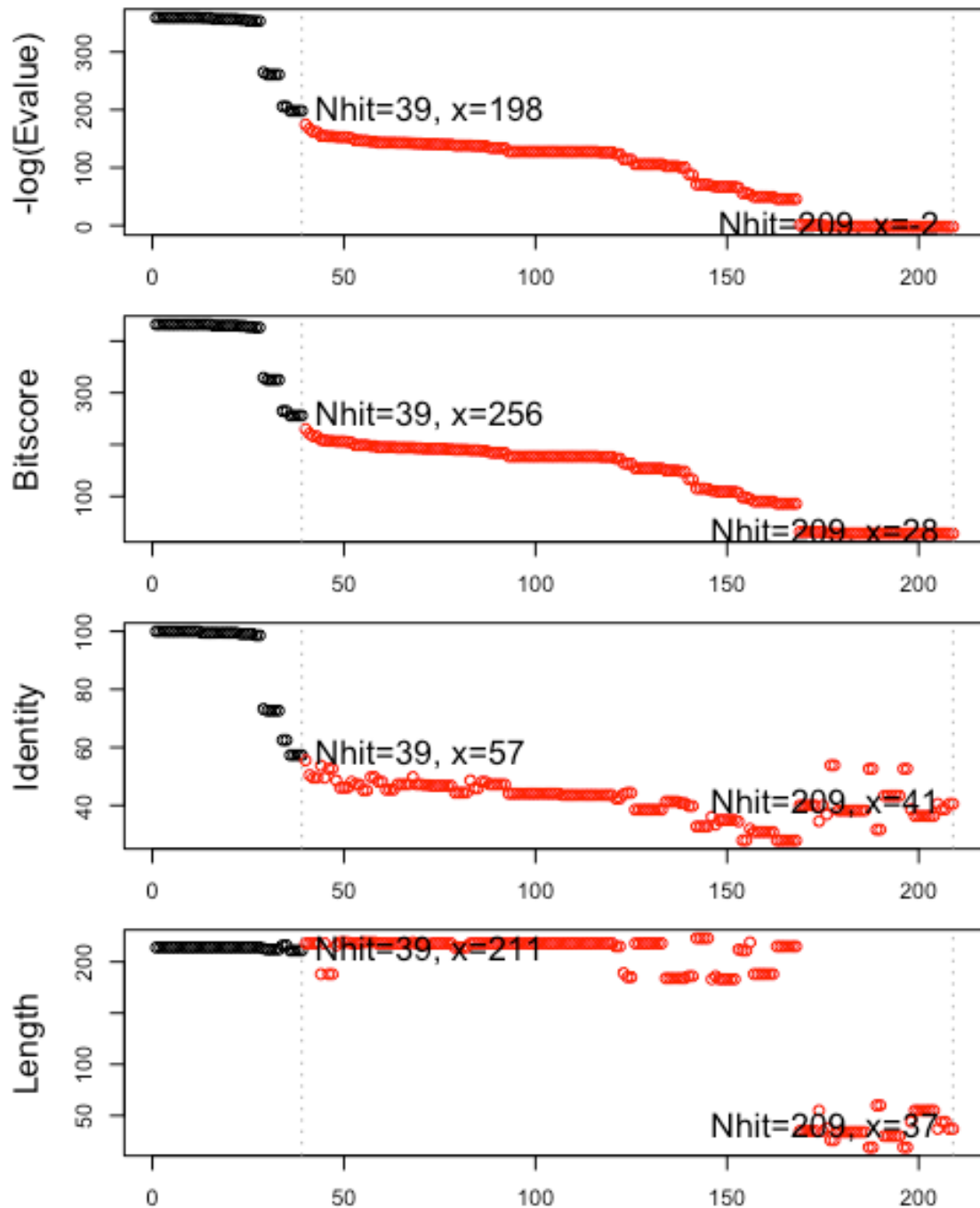


Figure 11: Blast results. Visualize and filter blast results through function `plot.blast()`. Here we proceed with only the top scoring hits (black).

```
head(hits$ pdb.id)
```

```
## [1] "1AKE_A" "1AKE_B" "1ANK_A" "1ANK_B" "4AKE_A" "4AKE_B"
```

The Blast search and subsequent filtering identified a total of 39 related PDB structures to our query sequence. The PDB identifiers of this collection are accessible through the `pdb.id` attribute to the `hits` object (`hits$ pdb.id`). Note that adjusting the `cutoff` argument (to `plot.blast()`) will result in a decrease or increase of hits.

We can now use function `get.pdb()` and `pdbslit()` to fetch and parse the identified structures. Finally, we use `pdbaln()` to align the PDB structures.

```
# Fetch PDBs
files <- get.pdb(hits$ pdb.id, path = "pdb", split = TRUE, gzip =
TRUE)

# Align structures
pdbslit(files)

# Vector containing PDB codes
ids <- basename.pdb(pdbslit$id)

# Draw schematic alignment
plot(pdbslit, labels=ids)
```

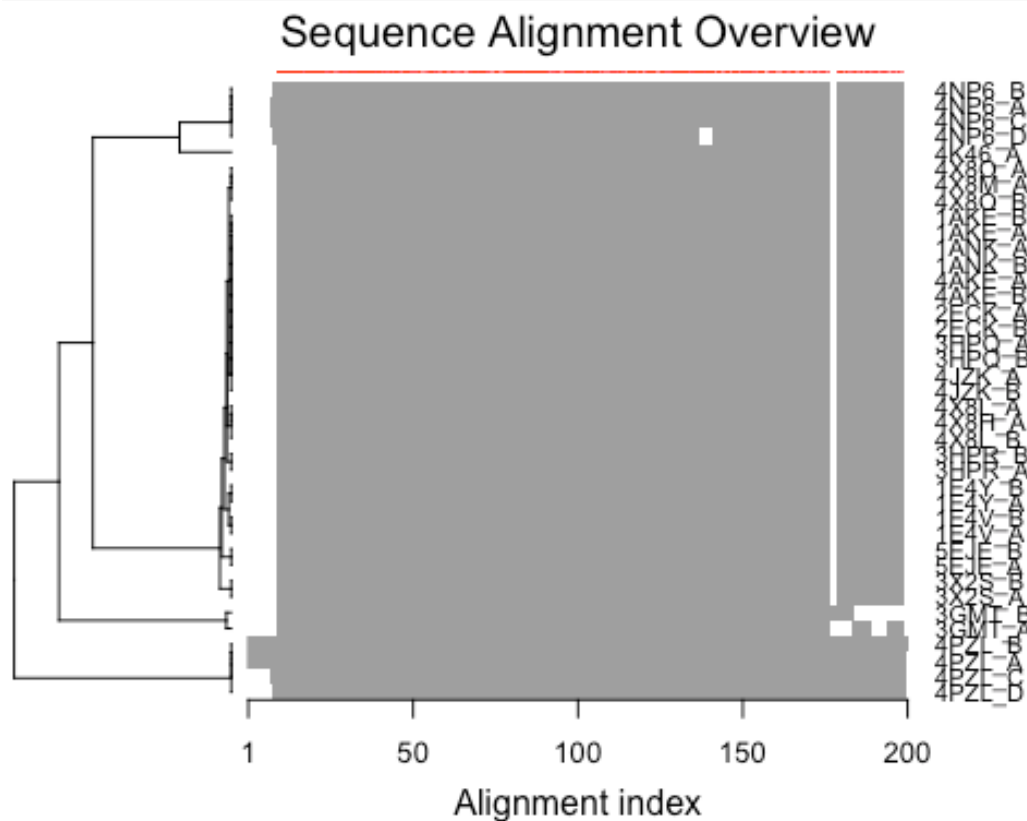


Figure 12: Schematic representation of alignment. Grey regions depict aligned residues, while white depict gap regions. The red bar at the top depict sequence conservation.

6.2 Sequence conservation analysis

```
# Calculate sequence conservation
cons <- conserv(pdbs, method="entropy22")

# SSE annotations
sse <- pdbs2sse(pdbs, ind=1, rm.gaps=FALSE)

# Plot conservation per residue
plotb3(cons, sse=sse, ylab="Sequence entropy")
```

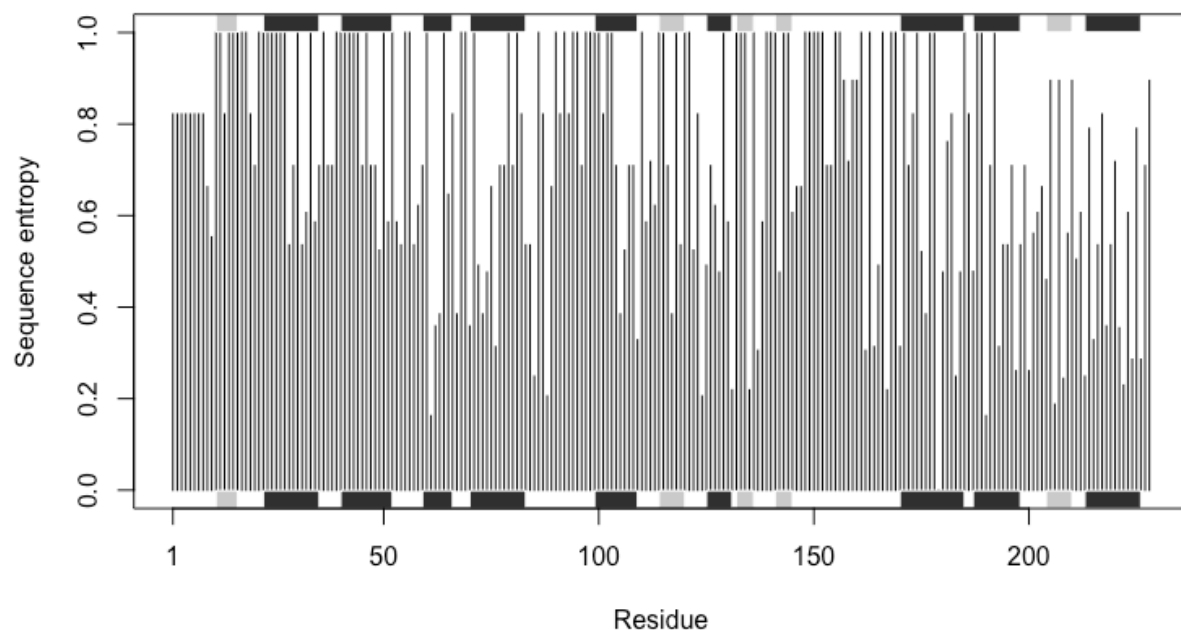


Figure 13: Sequence conservation per residue. Here, Shannon's information entropy is used to measure the diversity per alignment column. SSEs are depicted with dark (helices) and light (sheets) grey boxes in marginal regions.

6.3 Annotate collected PDB structures

Function `pdb.annotate()` provides a convenient way of annotating the PDB files we have collected. Below we use the function to annotate each structure to its source species. This will come in handy when annotating plots later on:

```
anno <- pdb.annotate(ids)
print(unique(anno$source))
```



```
## [1] "Escherichia coli"           "Photobacterium profundum"  
## [3] "Vibrio cholerae"            "Burkholderia pseudomallei"  
## [5] "Francisella tularensis"
```

6.4 Principal component analysis

A principal component analysis (PCA) can be performed on the structural ensemble (stored in the *pdb*s object) with function `pca.xyz()`. To obtain meaningful results we first superimpose all structures on the *invariant core* (function `core.find()`).

```
# find invariant core  
core <- core.find(pdb)  
  
# superimpose all structures to core  
pdb$xyz = pdffit(pdb, core)  
  
# Perform PCA  
pc.xray <- pca(pdb)
```

Function **rmsd()** will calculate all pairwise RMSD values of the structural ensemble. This facilitates clustering analysis based on the pairwise structural deviation:

```
# Calculate RMSD  
rd <- rmsd(pdb)  
  
# Structure-based clustering  
hc.rd <- hclust(dist(rd))  
grps.rd <- cutree(hc.rd, k=3)  
  
plot(pc.xray, 1:2, col="grey50", bg=grps.rd, pch=21, cex=1)
```

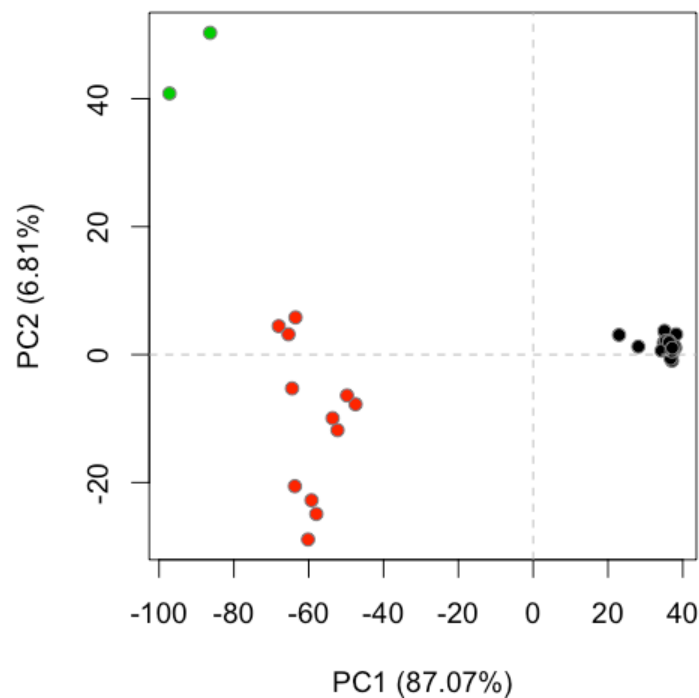


Figure 14: Projection of Adenylate kinase X-ray structures. Each dot represents one PDB structure.

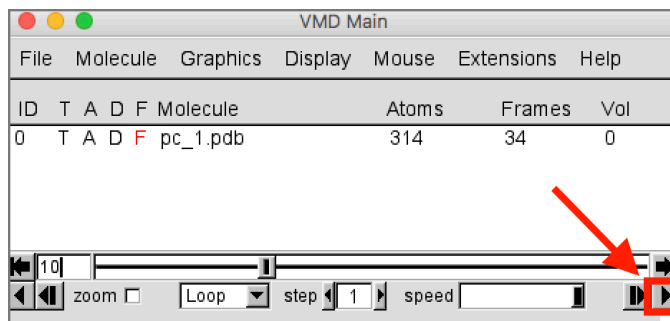
One can then use the `identify()` function to label and individual points.

```
# Left-click on a point to label and right-click to end
identify(pc.xray$z[,1:2], labels=basename.pdb(pdb$id))
```

To visualize the major structural variations in the ensemble the function **mktrj()** can be used to generate a trajectory PDB file by interpolating along the eigenvector:

```
# Visualize first principal component
mktrj(pc.xray, pc=1, file="pc_1.pdb")
```

You can open this file, `pc_1.pdb`, in VMD, chose the “Drawing Method” **Tube** and “Coloring Method” **Index**. Then click the play button shown below to animate the structure and visualize the major structural variations along PC1.



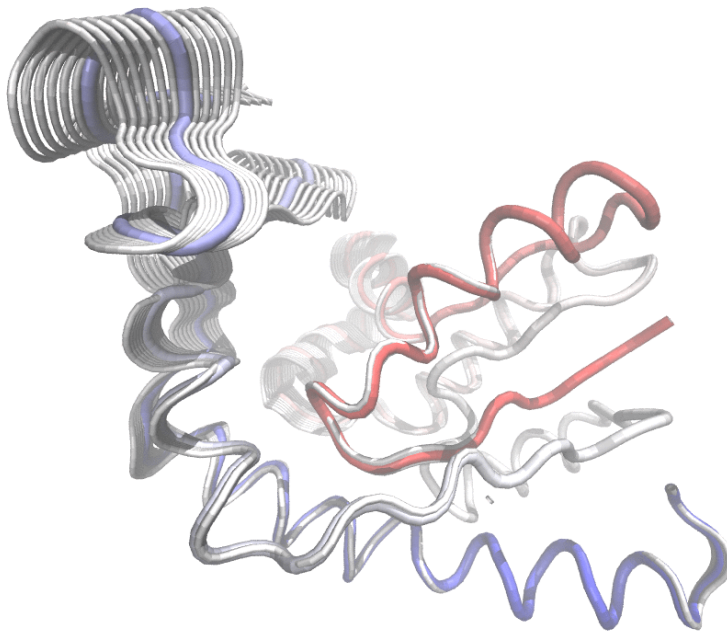


Figure 15: Visualization of PC-1 in VMD. Trajectory PDB file is generated using `mktrj()`.

6.5 Plotting results with ggplot2

```
library(ggplot2)
library(ggrepel)

## Warning: package 'ggrepel' was built under R version 3.4.2

df <- data.frame(x=pc.xray$z[,1], y=pc.xray$z[,2])
col <- as.factor(grps.rd)

p <- ggplot(df, aes(x, y)) +
  geom_point(aes(col=col), size=2) +
  xlab("PC1") +
  ylab("PC2") +
  scale_color_discrete(name="Clusters") +
  geom_text_repel(aes(label=ids))
p
```

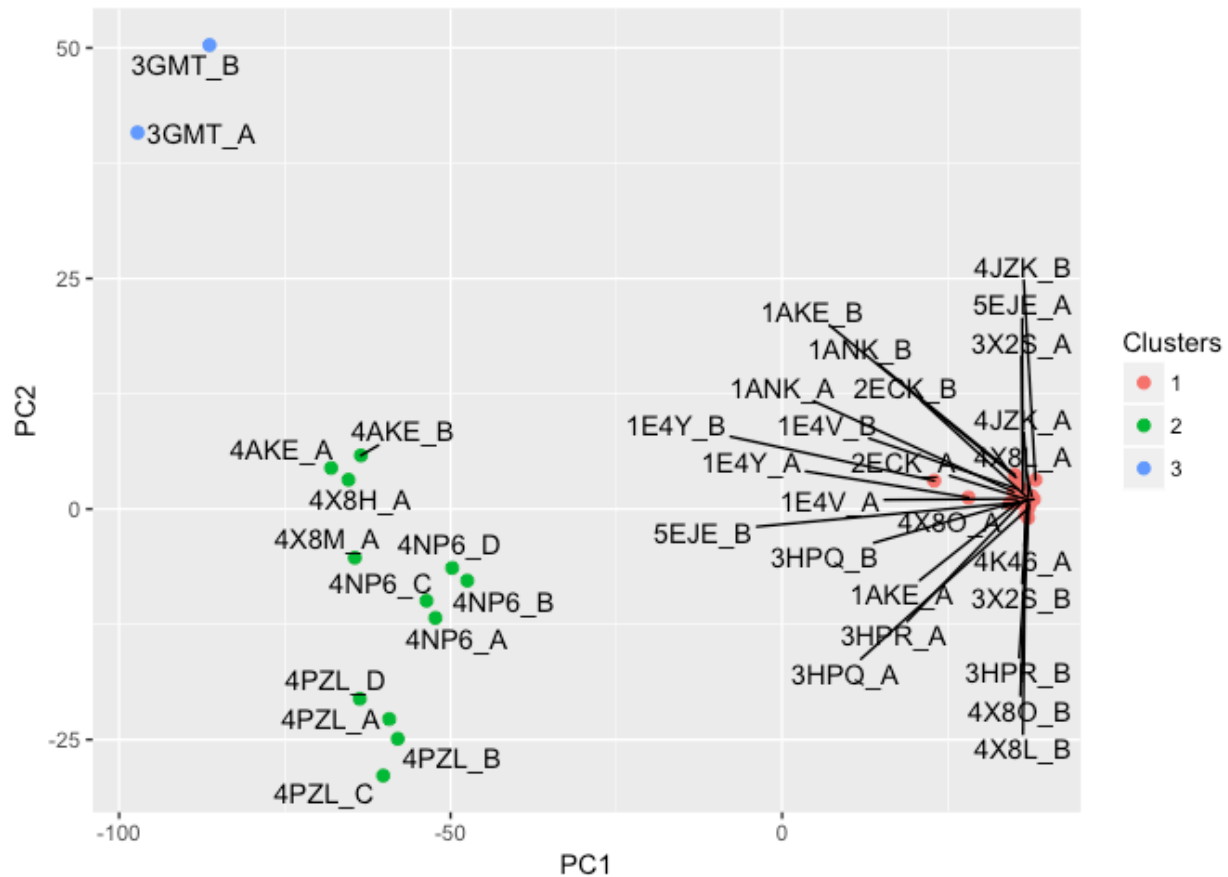


Figure 16: Projection of Adenylate kinase X-ray structures using package **ggplot2**. Each dot represents one PDB structure.

Q12. Muddy Point Assessment Feedback: How Please help us improve this lab session by providing your anonymous opinions here: <https://goo.gl/forms/elweVfSycVlt8Ecd2>

R Session info

```
# Information about the current Bio3D session
sessionInfo()

## R version 3.4.1 (2017-06-30)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Sierra 10.12.6
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.
```

```

0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/
libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods   base
##
## other attached packages:
## [1] ggrepel_0.7.0    ggplot2_2.2.1    bio3d_2.3-3.9000
##
## loaded via a namespace (and not attached):
## [1] Rcpp_0.12.13      knitr_1.17        magrittr_1.5      munsell_0.4.3
## [5] colorspace_1.3-2  rlang_0.1.2       stringr_1.2.0     highr_0.6
## [9] plyr_1.8.4        tools_3.4.1       parallel_3.4.1    grid_3.4.1
## [13] gtable_0.2.0      htmltools_0.3.6   yaml_2.1.14       lazyeval_0.2.0
## [17] rprojroot_1.2     digest_0.6.12     tibble_1.3.4      evaluate_0.10.1
## [21] rmarkdown_1.6     labeling_0.3       stringi_1.1.5     compiler_3.4.1
## [25] scales_0.5.0      backports_1.1.1

```

Section 7: Exploring the conformational dynamics of proteins with Bio3D-web

Visit the new web application Bio3D-web: <http://thegrantlab.org/bio3d/webapps> watch the introduction video and and click start analysis to begin exploring the conformational dynamics and flexibility of protein structures.