Structural Bioinformatics (Part 1)

Yash Garodia

15/02/2022

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

> Download a CSV file from the PDB site (accessible from "Analyze" >"PDB Statistics" > "by Experimental Method and Molecular Type". Move this CSV file into your RStudio project and use it to answer the following questions:
>
> Q1: What percentage of structures in the PDB are solved by X-Ray and Electron Microscopy.
>
> Q2: What proportion of structures in the PDB are protein?
>
> Q3: Type HIV in the PDB website search box on the home page and determine how many HIV-1 protease structures are in the current PDB?

Lets import the csv file

```
pdbstats <- read.csv("Data_Export_Summary.csv", row.names = 1)
pdbstats
```

```
##                          X.ray   NMR   EM Multiple.methods Neutron Other  Total
## Protein (only)          144433 11881 6732              182      70    32 163330
## Protein/Oligosaccharide   8543    31 1125                5       0     0   9704
## Protein/NA                7621   274 2165                3       0     0  10063
## Nucleic acid (only)       2396  1399   61                8       2     1   3867
## Other                      150    31    3                0       0     0    184
## Oligosaccharide (only)      11     6    0                1       0     4     22
```

Q1) To find the %structures in PDB solved by X-Ray:

```
#First we find out the total number of structures:
total <- sum(pdbstats[,7])
total
```

```
## [1] 187170
```

```
#Then we find the total number of structures solved by X-ray
total_xray <- sum(pdbstats[,1])
total_xray
```

```
## [1] 163154
```

```
#Then we find the total number of structures solved by EM:
total_em <- sum(pdbstats[,3])
total_em
```

```
## [1] 10086
```

```
#Finally we divide the total number of structures solved by X-ray by total number of structures, and mu
perc_xray <- total_xray/total * 100
perc_xray
```

```
## [1] 87.16888
```

87.17% of structures are solved by X-ray. Repeating the same procedure for Electron Microscopy,

```
total_em <- sum(pdbstats[,3])
total_em
```

```
## [1] 10086
```

```
perc_em <- total_em/total*100
perc_em
```

```
## [1] 5.388684
```

5.39% of structures are solved by Electron Microscopy.

Q2) To find what proportion of structures in the PDB are protein

```
# We divide the number of protein only structures by the total number of structures.
total_prot <- pdbstats[1,7]
total_prot/total
```

```
## [1] 0.8726292
```

Therefore, the proportion of structures in PDB that are protein is 0.873.

> Q3: Type HIV in the PDB website search box on the home page and determine how many HIV-1 protease structures are in the current PDB?

It is not easy to determine this from a text search for "HIV" or "HIV Protease" alone. A sequence search must be done to get much more reliable set of results.

2. Visualizing the HIV-1 protease structure

> Q4: Water molecules normally have 3 atoms. Why do we see just one atom per water molecule in this structure?

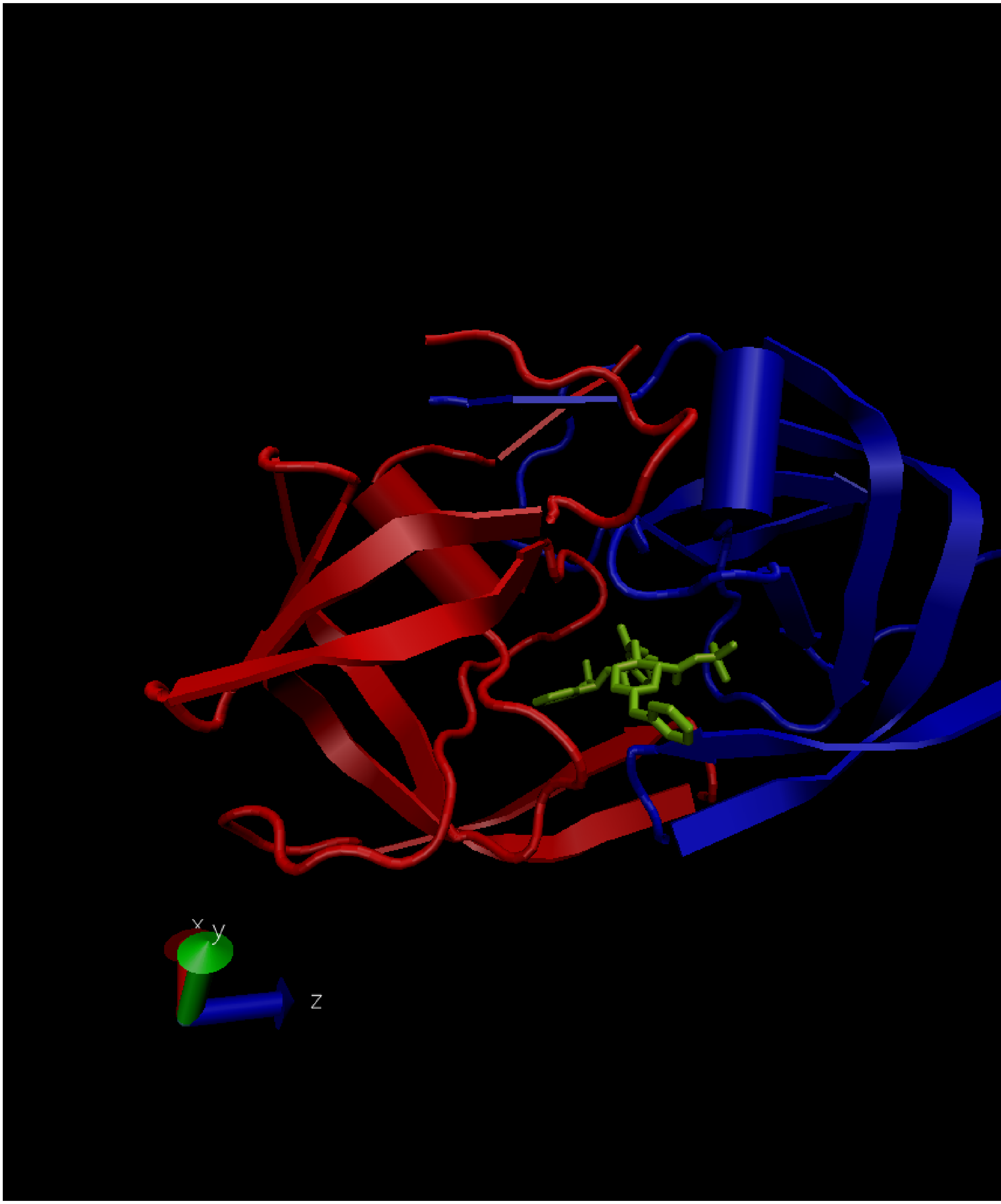
This is because a VDW viewer was used to see the water molecules, and van der waals forces don't exist within individual atoms in a water molecule (hydrogen bonds exist within each molecule); they only exist between different water molecules. Hence, each water molecule acts as a single point that faces van der waals forces from another water molecule.

> Q5: 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)?

This water molecule has residue #308.

> 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). Upload this figure to Piazza for some extra credit.

```r
knitr::include_graphics("class_11_optional.png")
```


x
y
z

Sent to instructors via piazza as a note as well(private note @103)

> 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?

Alpha helices.

3. Introduction to Bio3D in R

```r
library(bio3d)
pdb <- read.pdb("1hsg")
```

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

```r
#To get a quick summary of the contents of the pdb object you just created you can issue the command pr
pdb
```

```
##
##  Call:  read.pdb(file = "1hsg")
##
##    Total Models#: 1
##      Total Atoms#: 1686,  XYZs#: 5058  Chains#: 2  (values: A B)
##
##      Protein Atoms#: 1514  (residues/Calpha atoms#: 198)
##      Nucleic acid Atoms#: 0  (residues/phosphate atoms#: 0)
##
##      Non-protein/nucleic Atoms#: 172  (residues: 128)
##      Non-protein/nucleic resid values: [ HOH (127), MK1 (1) ]
##
##    Protein sequence:
##       PQITLWQRPLVTIKIGGQLKEALLDTGADDTVLEEMSLPGRWKPKMIGGIGGFIKVRQYD
##       QILIEICGHKAIGTVLVGPTPVNIIGRNLLTQIGCTLNFPQITLWQRPLVTIKIGGQLKE
##       ALLDTGADDTVLEEMSLPGRWKPKMIGGIGGFIKVRQYDQILIEICGHKAIGTVLVGPTP
##       VNIIGRNLLTQIGCTLNF
##
## + attr: atom, xyz, seqres, helix, sheet,
##         calpha, remark, call
```

> Q7: How many amino acid residues are there in this pdb object?

This is given by the "residues/Calpha atoms" section. There are 198 amino acid residues in this pdb object.

> Q8: Name one of the two non-protein residues?

HOH or water is one of the non protein residues. This is given by the "Non-protein/nucleic resid values" section.

> Q9: How many protein chains are in this structure?

There are two protein chains in this structure. The answer to this is in the "chains#" section.

> 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     b
## 1 ATOM     1     N <NA>   PRO     A     1   <NA> 29.361 39.686 5.862 1 38.10
## 2 ATOM     2    CA <NA>   PRO     A     1   <NA> 30.307 38.663 5.319 1 40.62
## 3 ATOM     3     C <NA>   PRO     A     1   <NA> 29.760 38.071 4.022 1 42.64
## 4 ATOM     4     O <NA>   PRO     A     1   <NA> 28.600 38.302 3.676 1 43.40
## 5 ATOM     5    CB <NA>   PRO     A     1   <NA> 30.508 37.541 6.342 1 37.87
## 6 ATOM     6    CG <NA>   PRO     A     1   <NA> 29.296 37.591 7.162 1 38.40
##   segid elesy charge
## 1  <NA>     N   <NA>
## 2  <NA>     C   <NA>
## 3  <NA>     C   <NA>
## 4  <NA>     O   <NA>
## 5  <NA>     C   <NA>
## 6  <NA>     C   <NA>
```

4. Comparative structure analysis of Adenylate Kinase

#Overview

> Starting from only one Adk PDB identifier (PDB ID: 1AKE) we will search the entire PDB for related structures using BLAST, fetch, align and superpose the identified structures, perform PCA and finally calculate the normal modes of each individual structure in order to probe for potential differences in structural flexibility.

Setup

```
# Install packages in the R console not your Rmd

#install.packages("bio3d")
#install.packages("ggplot2")
```

```
#install.packages("ggrepel")
#install.packages("devtools")
#install.packages("BiocManager")

#BiocManager::install("msa")
#devtools::install_bitbucket("Grantlab/bio3d-view")
```

> Q10) Which of the packages above is found only on BioConductor and not CRAN?

msa; this is a package we had to install from BioConductor(BiocManager)

> Q11) Which of the above packages is not found on BioConductor or CRAN?

bio3d-view. This had to be taken from the Grantlab website with the help of Devtools.

> Q12) True or False? Functions from the devtools package can be used to install packages from GitHub and BitBucket?

True

Search and retrieve ADK structures

> Below we perform a blast search of the PDB database to identify related structures to our query Adenylate kinase (ADK) 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 blast.pdb(). Note that get.seq() would also allow the corresponding UniProt identifier.

```
library(bio3d)
aa <- get.seq("1ake_A")
```

```
## Warning in get.seq("1ake_A"): Removing existing file: seqs.fasta
```

```
## Fetching... Please wait. Done.
```

```
aa
```

```
##                1        .         .         .         .         .         60
## pdb|1AKE|A   MRIILLGAPGAGKGTQAQFIMEKYGIPQISTGDMLRAAVKSGSELGKQAKDIMDAGKLVT
##                1        .         .         .         .         .         60
##
##               61        .         .         .         .         .         120
## pdb|1AKE|A   DELVIALVKERIAQEDCRNGFLLDGFPRTIPQADAMKEAGINVDYVLEFDVPDELIVDRI
##               61        .         .         .         .         .         120
##
##              121        .         .         .         .         .         180
## pdb|1AKE|A   VGRRVHAPSGRVYHVKFNPPKVEGKDDVTGEELTTRKDDQEETVRKRLVEYHQMTAPLIG
##              121        .         .         .         .         .         180
##
##              181        .         .         .   214
## pdb|1AKE|A   YYSKEAEAGNTKYAKVDGTKPVAEVRADLEKILG
##            181        .         .         .   214
##
## Call:
##   read.fasta(file = outfile)
##
## Class:
##   fasta
##
## Alignment dimensions:
##   1 sequence rows; 214 position columns (214 non-gap, 0 gap)
##
## + attr: id, ali, call
```

> Q13. How many amino acids are in this sequence, i.e. how long is this sequence?

There are 214 amino acids in this sequence.

> Now we can use this sequence as a query to BLAST search the PDB to find similar sequences and structures.

```
# Blast or hmmer search

b <- blast.pdb(aa)
```

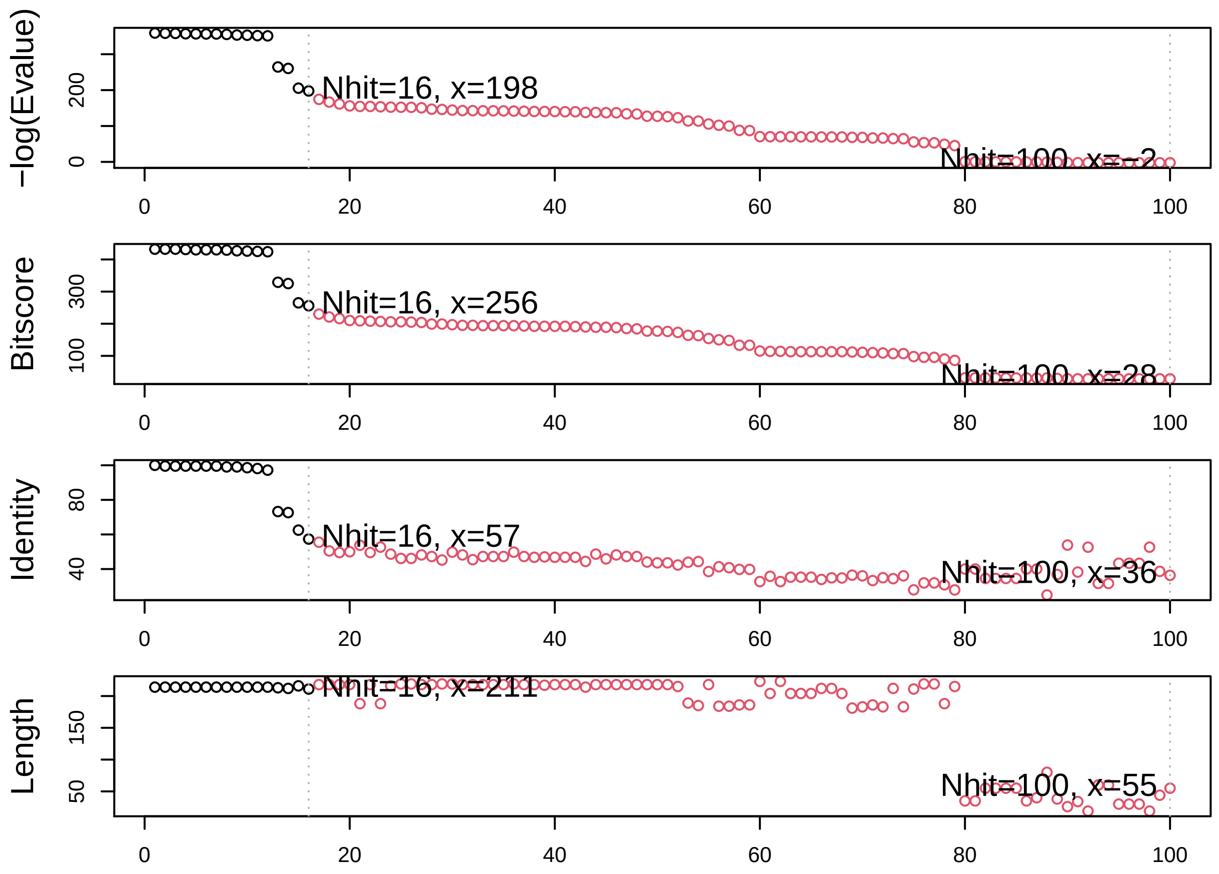
```
##  Searching ... please wait (updates every 5 seconds) RID = 0VVAJVB4016
##  .
##  Reporting 100 hits
```

> The 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 to include only the relevant E.coli structures:

```
# Plot a summary of search results

hits <- plot(b)
```

```
##   * Possible cutoff values:    197 -3
##             Yielding Nhits:    16 100
##
##   * Chosen cutoff value of:    197
##             Yielding Nhits:    16
```



```r
# List out some 'top hits'
head(hits$pdb.id)
```

```
## [1] "1AKE_A" "4X8M_A" "6S36_A" "6RZE_A" "4X8H_A" "3HPR_A"
```

The Blast search and subsequent filtering identified a total of 16 related PDB structures to our query sequence. The PDB identifiers of this collection are accessible through the $pdb.id attribute to the hits object (i.e. hits$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.

```r
#Download related PDB files

files <- get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T)
```

```
## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 1AKE.pdb.gz exists. Skipping download

## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 4X8M.pdb.gz exists. Skipping download

## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 6S36.pdb.gz exists. Skipping download
## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 6RZE.pdb.gz exists. Skipping download
```

```
## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 4X8H.pdb.gz exists. Skipping download
```

```
## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 3HPR.pdb.gz exists. Skipping download
```

```
## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 1E4V.pdb.gz exists. Skipping download
```

```
## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 5EJE.pdb.gz exists. Skipping download
```

```
## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 1E4Y.pdb.gz exists. Skipping download
```

```
## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 3X2S.pdb.gz exists. Skipping download
```

```
## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 6HAP.pdb.gz exists. Skipping download
```

```
## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 6HAM.pdb.gz exists. Skipping download
```

```
## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 4K46.pdb.gz exists. Skipping download
```

```
## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 4NP6.pdb.gz exists. Skipping download
```

```
## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 3GMT.pdb.gz exists. Skipping download
```

```
## Warning in get.pdb(hits$pdb.id, path = "pdbs", split = T, gzip = T): pdbs/
## 4PZL.pdb.gz exists. Skipping download
```

```
##   |                                                                          |
```

Align and superpose structures

Next we will use the pdbaln() function to align and also optionally fit (i.e. superpose) the identified PDB structures.

```
# Align releated PDBs
pdbs <- pdbaln(files, fit = TRUE, exefile = "msa")
```

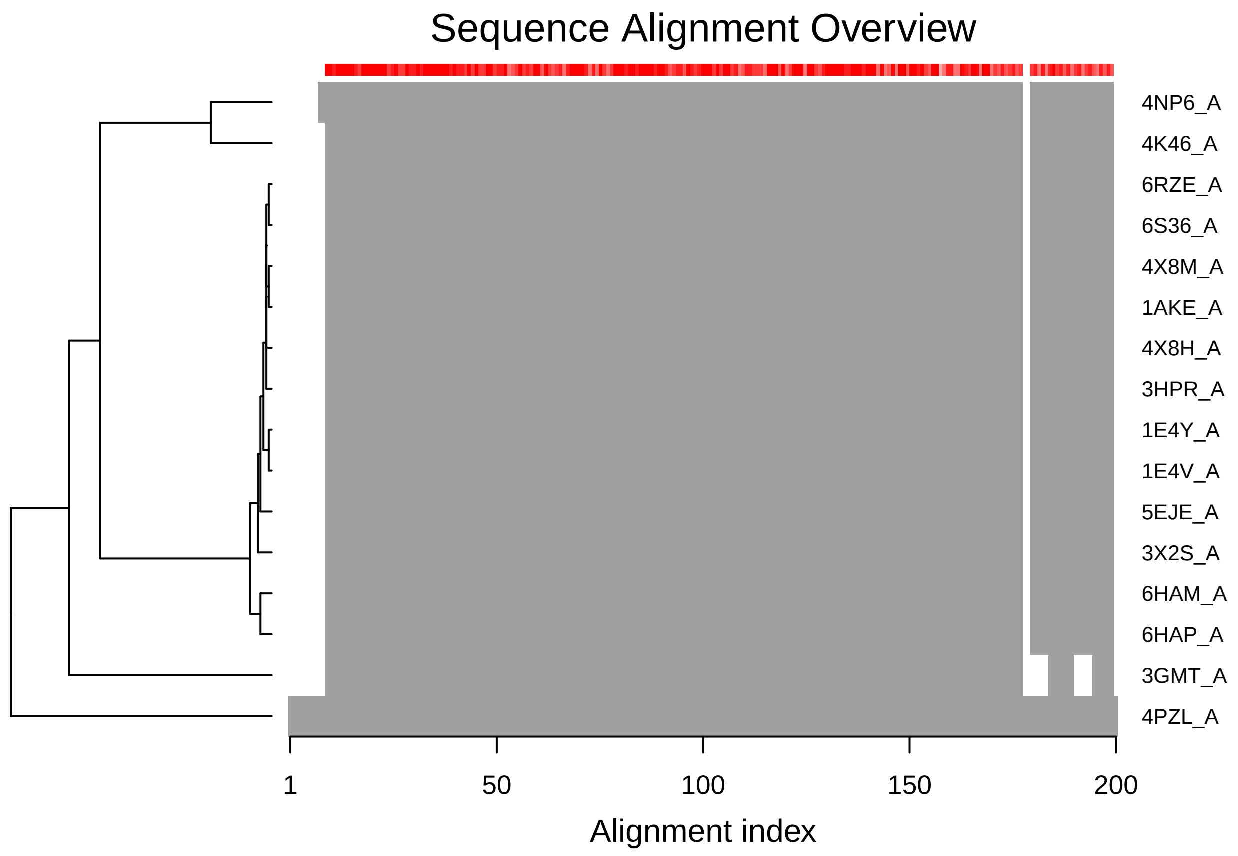
```
## Reading PDB files:
## pdbs/split_chain/1AKE_A.pdb
## pdbs/split_chain/4X8M_A.pdb
## pdbs/split_chain/6S36_A.pdb
## pdbs/split_chain/6RZE_A.pdb
## pdbs/split_chain/4X8H_A.pdb
## pdbs/split_chain/3HPR_A.pdb
## pdbs/split_chain/1E4V_A.pdb
## pdbs/split_chain/5EJE_A.pdb
## pdbs/split_chain/1E4Y_A.pdb
## pdbs/split_chain/3X2S_A.pdb
## pdbs/split_chain/6HAP_A.pdb
## pdbs/split_chain/6HAM_A.pdb
## pdbs/split_chain/4K46_A.pdb
## pdbs/split_chain/4NP6_A.pdb
## pdbs/split_chain/3GMT_A.pdb
## pdbs/split_chain/4PZL_A.pdb
##    PDB has ALT records, taking A only, rm.alt=TRUE
## ..   PDB has ALT records, taking A only, rm.alt=TRUE
## .   PDB has ALT records, taking A only, rm.alt=TRUE
## ..   PDB has ALT records, taking A only, rm.alt=TRUE
## ..   PDB has ALT records, taking A only, rm.alt=TRUE
## ....   PDB has ALT records, taking A only, rm.alt=TRUE
## .   PDB has ALT records, taking A only, rm.alt=TRUE
## ....
##
## Extracting sequences
##
## pdb/seq: 1   name: pdbs/split_chain/1AKE_A.pdb
##    PDB has ALT records, taking A only, rm.alt=TRUE
## pdb/seq: 2   name: pdbs/split_chain/4X8M_A.pdb
## pdb/seq: 3   name: pdbs/split_chain/6S36_A.pdb
##    PDB has ALT records, taking A only, rm.alt=TRUE
## pdb/seq: 4   name: pdbs/split_chain/6RZE_A.pdb
##    PDB has ALT records, taking A only, rm.alt=TRUE
## pdb/seq: 5   name: pdbs/split_chain/4X8H_A.pdb
## pdb/seq: 6   name: pdbs/split_chain/3HPR_A.pdb
##    PDB has ALT records, taking A only, rm.alt=TRUE
## pdb/seq: 7   name: pdbs/split_chain/1E4V_A.pdb
## pdb/seq: 8   name: pdbs/split_chain/5EJE_A.pdb
##    PDB has ALT records, taking A only, rm.alt=TRUE
## pdb/seq: 9   name: pdbs/split_chain/1E4Y_A.pdb
## pdb/seq: 10   name: pdbs/split_chain/3X2S_A.pdb
## pdb/seq: 11   name: pdbs/split_chain/6HAP_A.pdb
## pdb/seq: 12   name: pdbs/split_chain/6HAM_A.pdb
##    PDB has ALT records, taking A only, rm.alt=TRUE
## pdb/seq: 13   name: pdbs/split_chain/4K46_A.pdb
##    PDB has ALT records, taking A only, rm.alt=TRUE
## pdb/seq: 14   name: pdbs/split_chain/4NP6_A.pdb
## pdb/seq: 15   name: pdbs/split_chain/3GMT_A.pdb
## pdb/seq: 16   name: pdbs/split_chain/4PZL_A.pdb
```

```
#vector containing pdb codes for figure axis
```

```
ids <- basename.pdb(pdbs$id)

#draw schematic adjustment

plot(pdbs, labels = ids)
```



Optional: Viewing our superposed structures

We can view our superposed results with the new bio3d.view view() function:

```
library(bio3d)
library(devtools)
```

```
## Warning: package 'devtools' was built under R version 4.0.2

## Loading required package: usethis

## Warning: package 'usethis' was built under R version 4.0.2
```

```
library(bio3d.view)
library(rgl)
```

```
## Warning: package 'rgl' was built under R version 4.0.2
```

```
view.pdbs(pdbs)
```

Annotate collected PDB structures [Optional]

> The 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(c("2mh3_A", "4f3l"), anno.terms = c("structureId", "experimentalTechnique", "resolu
unique(anno$source)
```

```
## [1] "Homo sapiens" "Mus musculus"
```

> We can view all available annotation data:

```
anno
```

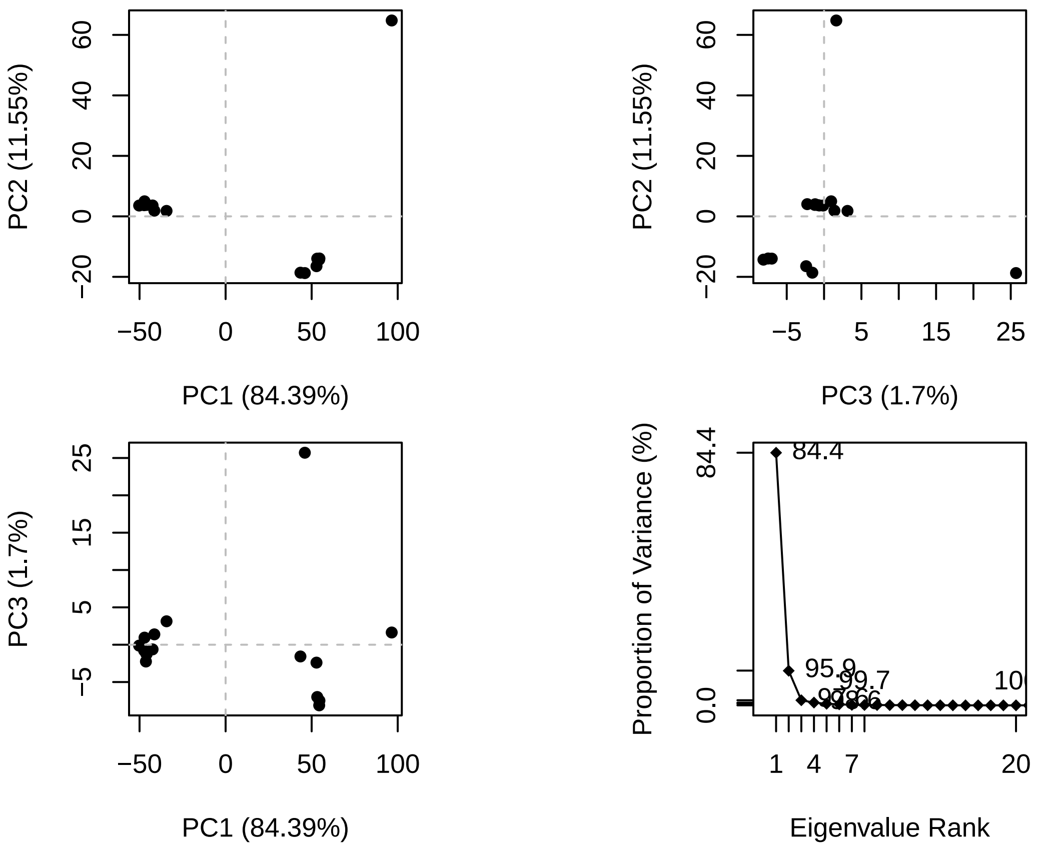
```
##        structureId experimentalTechnique resolution
## 2MH3_A        2MH3                   NMR         NA
## 4F3L_B        4F3L                 X-ray      2.268
## 4F3L_A        4F3L                 X-ray      2.268
##                                               pfam       source
## 2MH3_A Helix-loop-helix DNA-binding domain (HLH) Homo sapiens
## 4F3L_B                      PAS domain (PAS_11) Mus musculus
## 4F3L_A                      PAS domain (PAS_11) Mus musculus
##                                   citation
## 2MH3_A Popovic, M., et al. Proteins (2014)
## 4F3L_B    Huang, N., et al. Science (2012)
## 4F3L_A    Huang, N., et al. Science (2012)
```

This section didnt work as well. The response I got was “Error in split.default(X, group) : first argument must be a vector”

#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. PCA can be performed on the structural ensemble (stored in the pdbs object) with the function pca.xyz(), or more simply pca().

```
pc.xray <- pca(pdbs)
plot(pc.xray)
```



> Results of PCA on Adenylate kinase X-ray structures. Each dot represents one PDB structure.

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

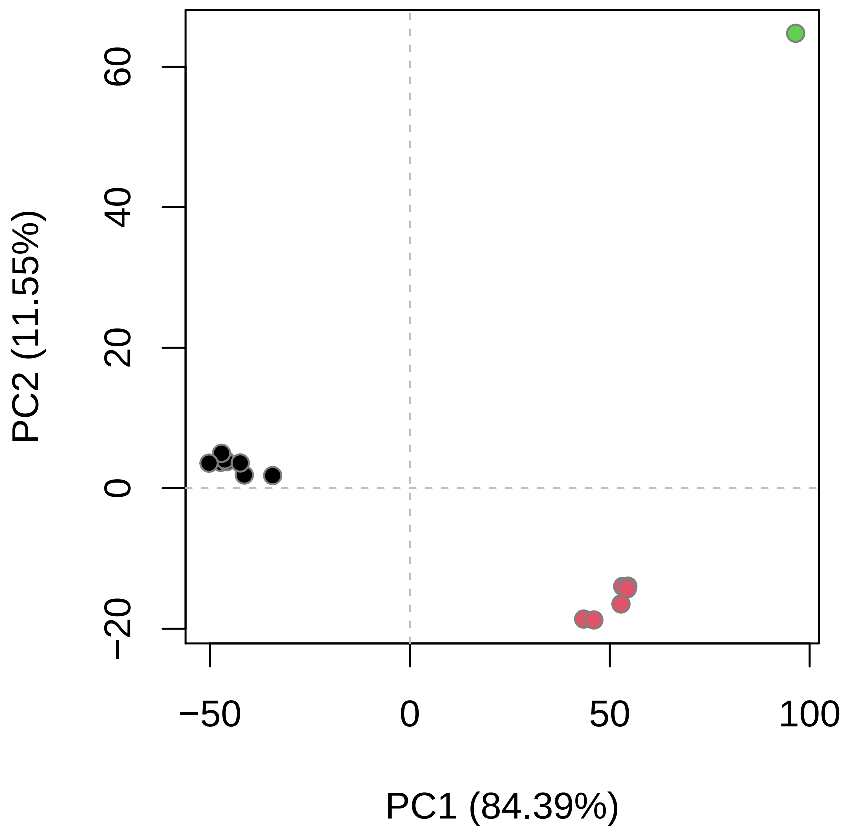
```
#calculate RMSD

rd <- rmsd(pdbs)
```

```
## Warning in rmsd(pdbs): No indices provided, using the 204 non NA positions
```

```
#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 10: Projection of Adenylate kinase X-ray structures. Each dot represents one PDB structure.

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. Optional further visualization

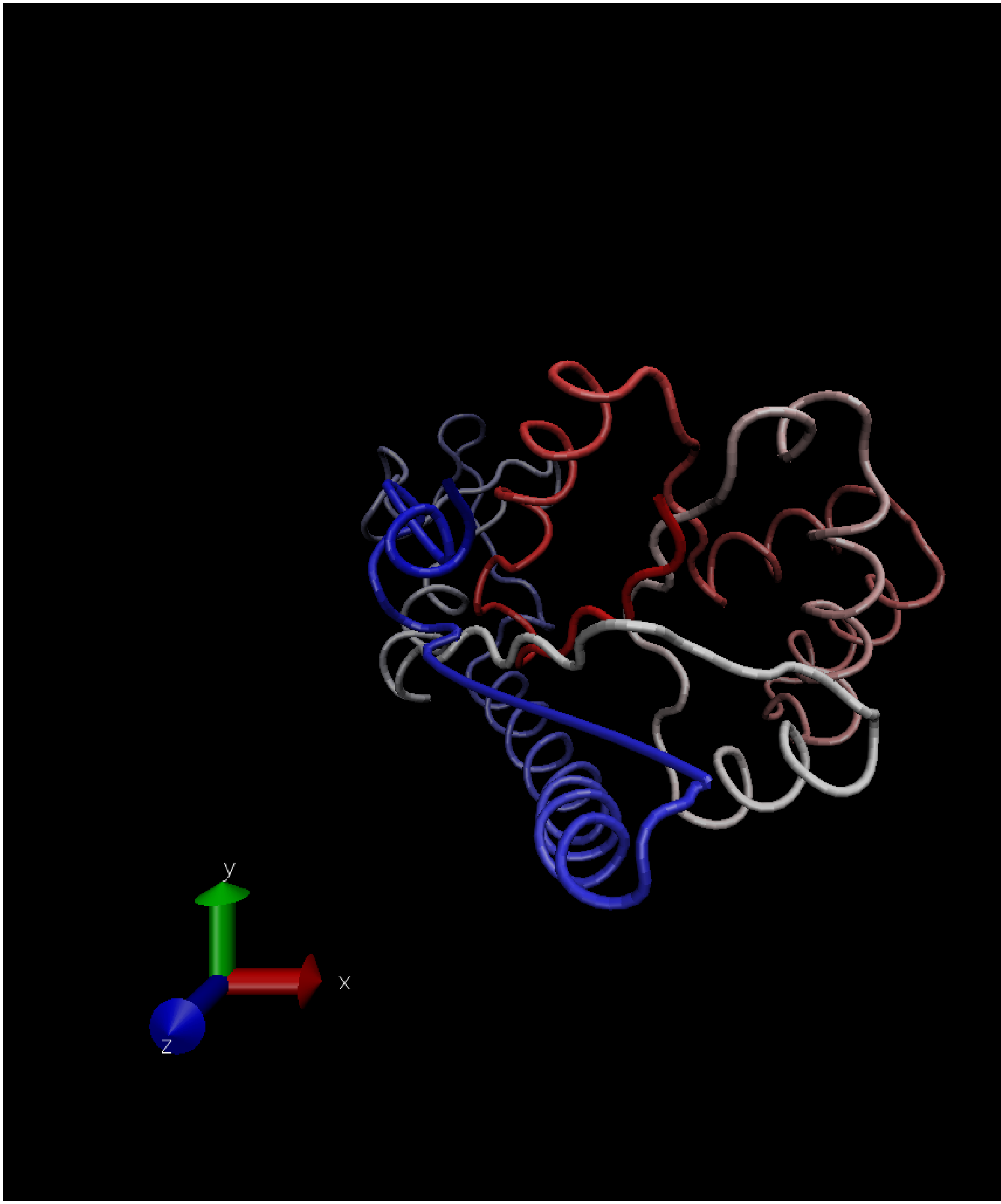
To visualize the major structural variations in the ensemble the function mktrj() can be used to generate a trajectory PDB file by interpolating along a give PC (eigenvector):

```r
#visualize the first principal component

pc1 <- 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.

```r
knitr::include_graphics("vmdscene.png")
```


y
x
z

> We can also view our results with the new bio3d.view view() function:

```
view.xyz(pc1)
```

```
## Potential all C-alpha atom structure(s) detected: Using calpha.connectivity()
```

We could set the color to highlight the most variable regions like so:

```
view.xyz(pc1, col=vec2color( rmsf(pc1) ))
```

```
## Potential all C-alpha atom structure(s) detected: Using calpha.connectivity()
```

We can also plot our main PCA results with ggplot:

```
#plotting results with ggplot2

library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.2
```

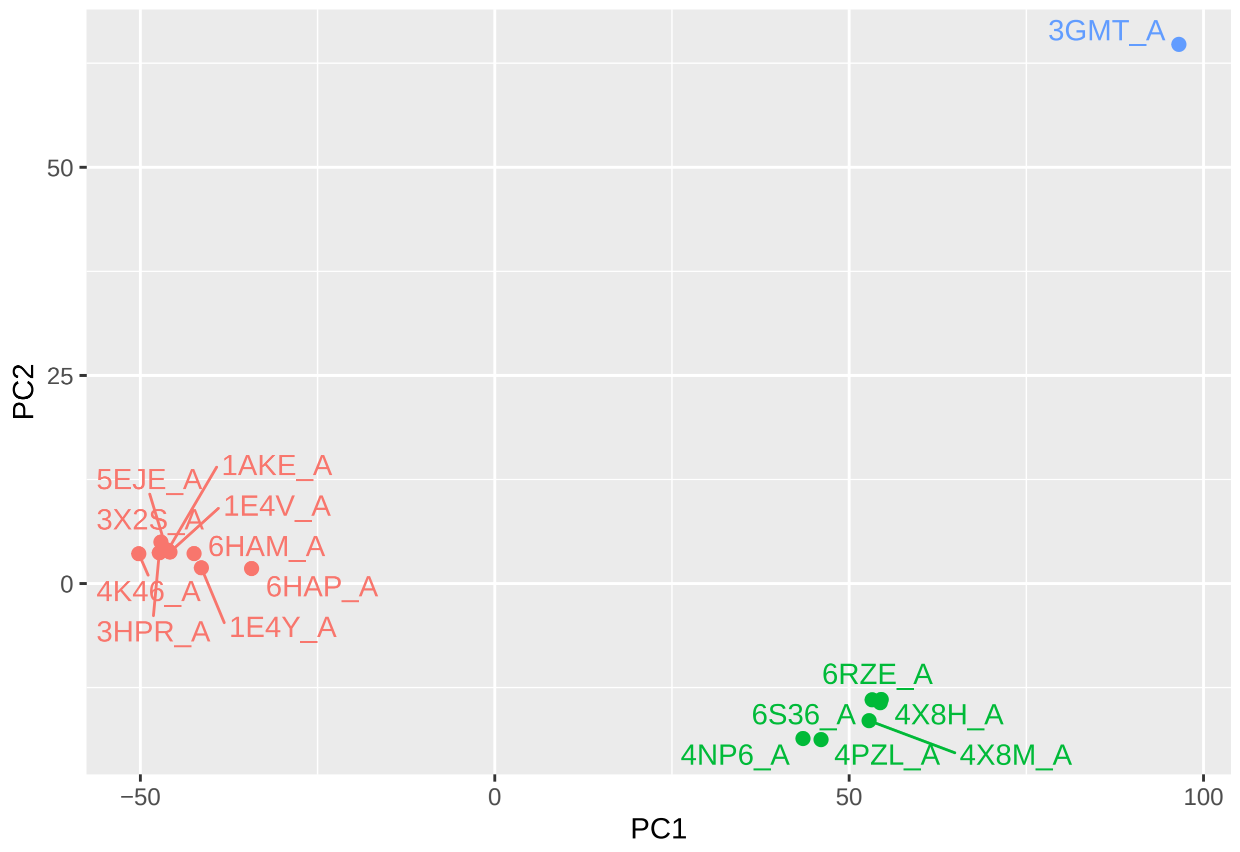
```
library(ggrepel)
```

```
## Warning: package 'ggrepel' was built under R version 4.0.2
```

```
df <- data.frame(PC1 = pc.xray$z[,1], PC2 = pc.xray$z[,2], col = as.factor(grps.rd), ids = ids)

p <- ggplot(df) + aes(PC1, PC2, col = col, label = ids) + geom_point(size = 2) + geom_text_repel(max.ove

p
```



6. Normal Mode Analysis

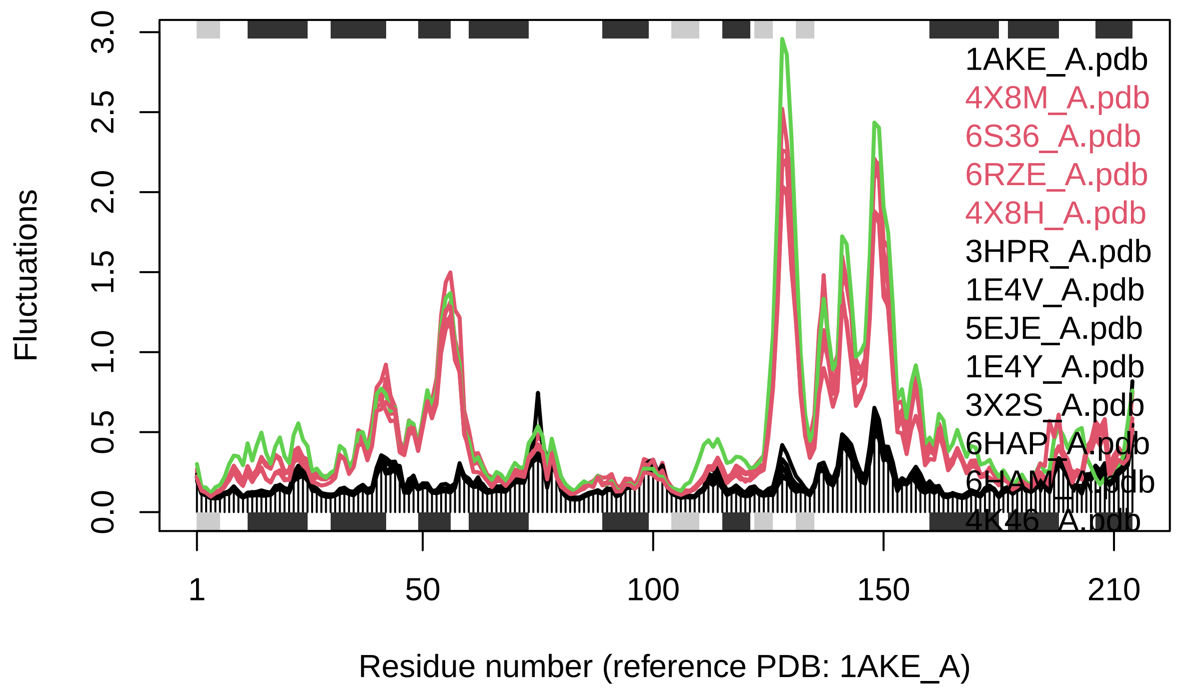
Function nma() provides normal mode analysis (NMA) on both single structures (if given a singe PDB input object) or the complete structure ensemble (if provided with a PDBS input object). This facilitates characterizing and comparing flexibility profiles of related protein structures.

```
# NMA of all structures
modes <- nma(pdbs)
```

```
##
## Details of Scheduled Calculation:
##   ... 16 input structures
##   ... storing 606 eigenvectors for each structure
##   ... dimension of x$U.subspace: ( 612x606x16 )
##   ... coordinate superposition prior to NM calculation
##   ... aligned eigenvectors (gap containing positions removed)
##   ... estimated memory usage of final 'eNMA' object: 45.4 Mb
##
##   |                                                                      |
```

```
plot(modes, pdbs, col = grps.rd)
```

```
## Extracting SSE from pdbs$sse attribute
```



> Q14) What do you note about this plot? Are the black and colored lines similar or different? Where do you think they differ most and why?

The black and colored lines are very different, with the black lines having a shorter peak and lesser fluctuations. They differ the most in the residue number range of 125-160 and 25-75 and it is due to the fact that these are the residues involved in the binding site of the protein; upon ligand binding, the protein changes conformational state. This suggests that there are 2 binding sites on the protein as well.

> Collectively these results indicate the existence of two major distinct conformational states for Adk. These differ by a collective low frequency displacement of two nucleotide-binding site regions that display distinct flexibilities upon nucleotide binding.

```
sessionInfo()
```

```
## R version 4.0.0 (2020-04-24)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS  10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/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_0.9.1         ggplot2_3.3.5         rgl_0.108.3
## [4] bio3d.view_0.1.0.9000 devtools_2.4.3        usethis_2.1.5
## [7] bio3d_2.4-3.9000
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.8          prettyunits_1.1.1   png_0.1-7
##  [4] ps_1.6.0            Biostrings_2.56.0   assertthat_0.2.1
##  [7] rprojroot_2.0.2     digest_0.6.28       utf8_1.2.2
## [10] R6_2.5.1            stats4_4.0.0        evaluate_0.14
## [13] httr_1.4.2          highr_0.9           pillar_1.6.3
## [16] zlibbioc_1.34.0     rlang_0.4.11        curl_4.3.2
## [19] rstudioapi_0.13     callr_3.7.0         S4Vectors_0.26.1
## [22] rmarkdown_2.11      labeling_0.4.2      desc_1.4.0
## [25] stringr_1.4.0       htmlwidgets_1.5.4   munsell_0.5.0
## [28] compiler_4.0.0      xfun_0.29           pkgconfig_2.0.3
## [31] BiocGenerics_0.34.0 pkgbuild_1.2.0      htmltools_0.5.2
## [34] tidyselect_1.1.1    tibble_3.1.5        IRanges_2.22.2
## [37] fansi_0.5.0         crayon_1.4.1        dplyr_1.0.7
## [40] withr_2.4.2         grid_4.0.0          DBI_1.1.1
## [43] jsonlite_1.7.2      gtable_0.3.0        lifecycle_1.0.1
## [46] magrittr_2.0.1      scales_1.1.1        cli_3.2.0
## [49] stringi_1.7.5       cachem_1.0.6        farver_2.1.0
## [52] XVector_0.28.0      fs_1.5.2            remotes_2.4.2
## [55] testthat_3.1.0      generics_0.1.0      ellipsis_0.3.2
## [58] vctrs_0.3.8         tools_4.0.0         msa_1.20.1
## [61] glue_1.6.1          purrr_0.3.4         processx_3.5.2
## [64] pkgload_1.2.2       parallel_4.0.0      fastmap_1.1.0
## [67] yaml_2.2.1          colorspace_2.0-2    sessioninfo_1.2.2
## [70] memoise_2.0.0       knitr_1.36
```