

hw06

Rachel Field

```
# Can you improve this analysis code?  
library(bio3d)  
s1 <- read.pdb("4AKE") # kinase with drug
```

Note: Accessing on-line PDB file

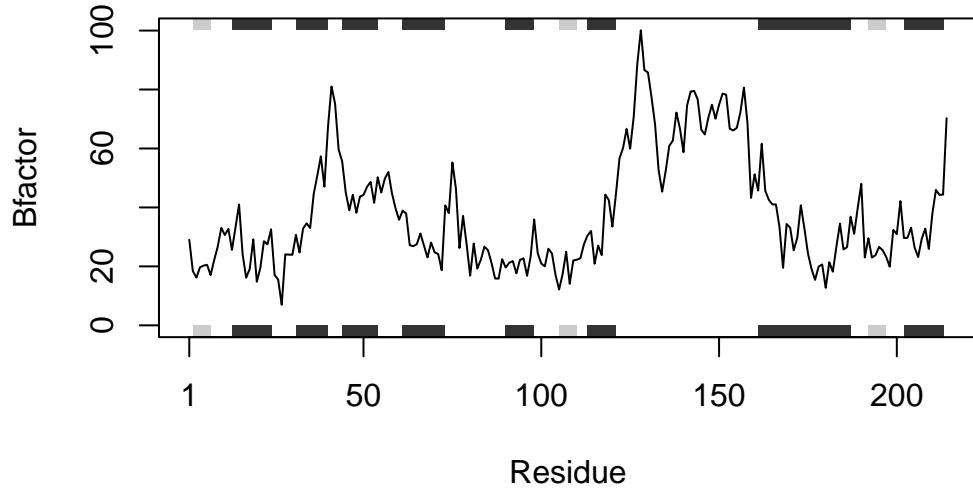
```
s2 <- read.pdb("1AKE") # kinase no drug
```

Note: Accessing on-line PDB file
PDB has ALT records, taking A only, rm.alt=TRUE

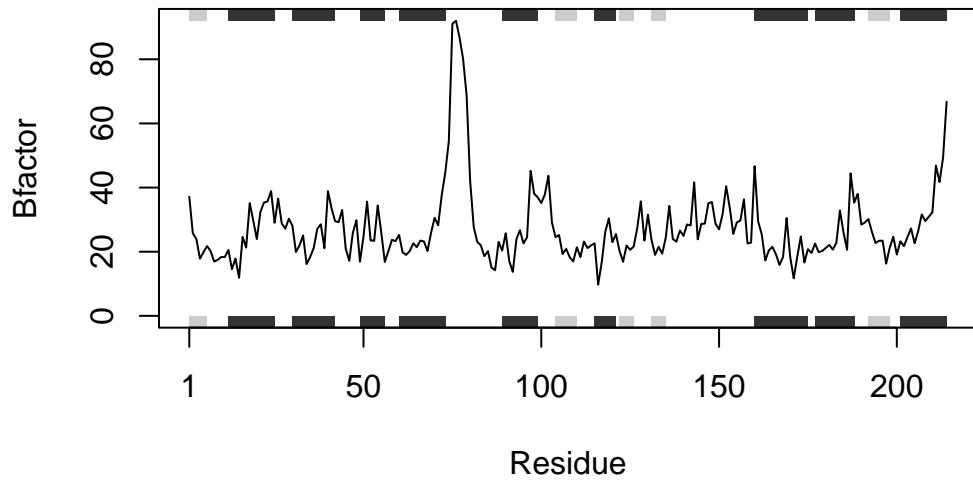
```
s3 <- read.pdb("1E4Y") # kinase with drug
```

Note: Accessing on-line PDB file

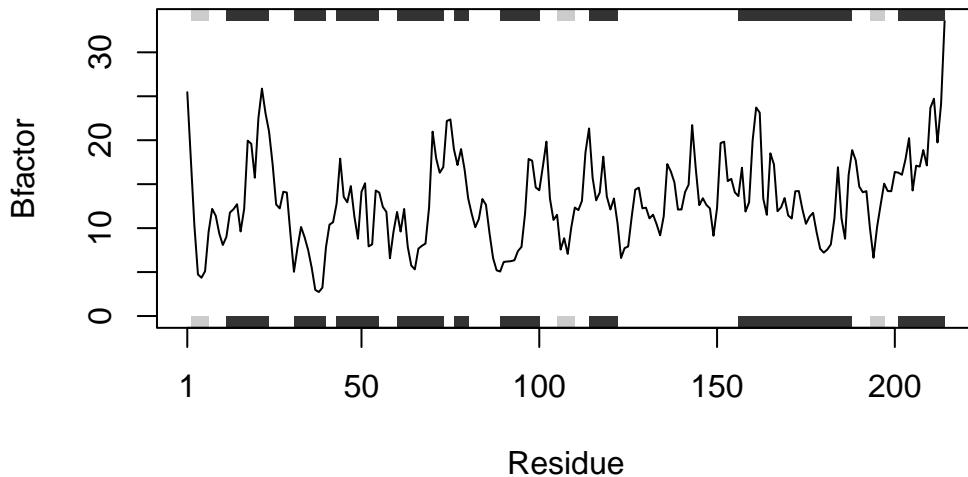
```
s1.chainA <- trim.pdb(s1, chain="A", elety="CA")  
s2.chainA <- trim.pdb(s2, chain="A", elety="CA")  
s3.chainA <- trim.pdb(s3, chain="A", elety="CA")  
s1.b <- s1.chainA$atom$b  
s2.b <- s2.chainA$atom$b  
s3.b <- s3.chainA$atom$b  
plotb3(s1.b, sse=s1.chainA, typ="l", ylab="Bfactor")
```



```
plotb3(s2.b, sse=s2.chainA, typ="l", ylab="Bfactor")
```



```
plotb3(s3.b, sse=s3.chainA, typ="l", ylab="Bfactor")
```



#Q1. What type of object is returned from the read.pdb() function?

A data frame

#Q2. What does the trim.pdb() function do?

It trims the full PDB data to include only atoms that meet specific criteria, like a specific chain (i.e. "A") or a specific atom type (i.e. "CA").

#Q3. What input parameter would turn off the marginal black and grey rectangles in the plots and what do they represent in this case?

The marginal black and grey rectangles are secondary structure elements (sses). The black rectangles are alpha helices and the grey rectangles are beta sheets. To turn the rectangles off, use sse=NULL in the plot functions.

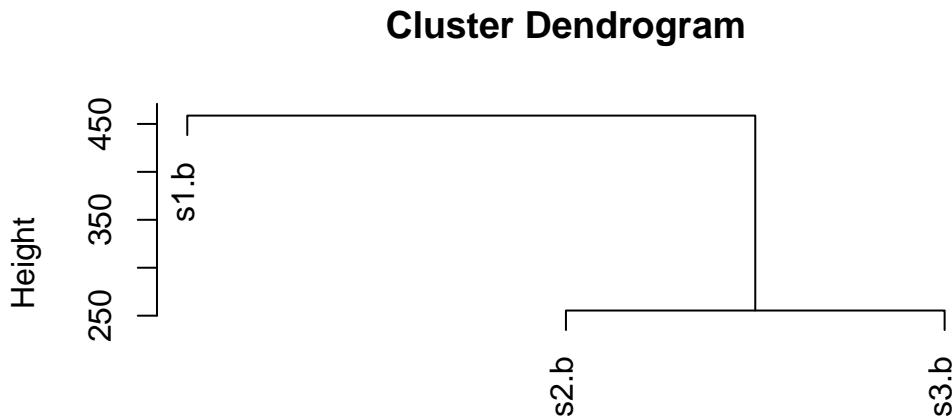
#Q4. What would be a better plot to compare across the different proteins?

One plot combining all of the three plots overlaid on the same x-y plot, using three colors to differentiate the original three plots, would be better to compare across the different proteins.

#Q5. Which proteins are more similar to each other in their B-factor trends. How could you quantify this? HINT: try the rbind(), dist() and hclust() functions together with a resulting dendrogram plot. Look up the documentation to see what each of these functions does.

s1.b and s3.b are the most similar to each other in their B-factor trends.

```
hc <- hclust( dist( rbind(s1.b, s2.b, s3.b) ) )
plot(hc)
```



```
dist(rbind(s1.b, s2.b, s3.b))
hclust (*, "complete")
```

#Q6. How would you generalize the original code above to work with any set of input protein structures?

```
library(bio3d)

any_char_vecs_pdbs <- c("4AKE", "1AKE", "1E4Y")

analyze_proteins <- function(any_char_vecs_pdbs) {
  trimmed_list <- lapply(any_char_vecs_pdbs, function(single_pdb) {
    pdb <- read.pdb(single_pdb)
    trim.pdb(pdb, chain = "A", elety = "CA")
  })

  beta_factors <- lapply(trimmed_list, function(pdb) pdb$atom$b)

  plotb3(beta_factors[[1]], sse = trimmed_list[[1]], typ = "l", ylab = "B-factor", main = "B-fa
  if(length(beta_factors)>1) {
```

```
for(i in 2:length(beta_factors)) {  
  lines(beta_factors[[i]], col = i)  
}  
}  
}  
  
analyze_proteins(any_char_vecs_pdbs)
```

Note: Accessing on-line PDB file

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):  
/var/folders/y1/t57v_5zd4kxc19_jdfkr2tpm0000gn/T//RtmpoQKbAL/4AKE.pdb exists.  
Skipping download
```

Note: Accessing on-line PDB file

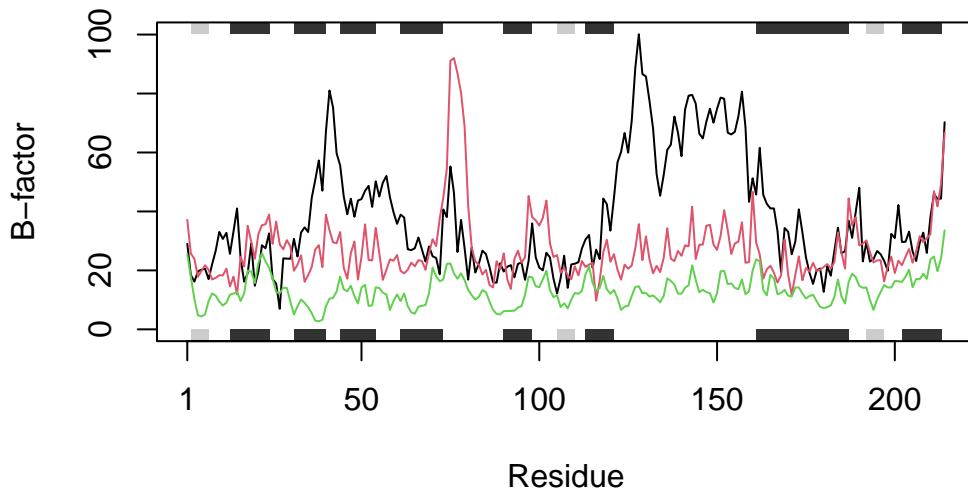
```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):  
/var/folders/y1/t57v_5zd4kxc19_jdfkr2tpm0000gn/T//RtmpoQKbAL/1AKE.pdb exists.  
Skipping download
```

PDB has ALT records, taking A only, rm.alt=TRUE

Note: Accessing on-line PDB file

```
Warning in get.pdb(file, path = tempdir(), verbose = FALSE):  
/var/folders/y1/t57v_5zd4kxc19_jdfkr2tpm0000gn/T//RtmpoQKbAL/1E4Y.pdb exists.  
Skipping download
```

B-factor Comparison



First, I loaded the bio3d package, which lets me use functions like `read.pdb()`, `trim.pdb()`, and `plotb3()` to load, change, and plot the PDB data.

Then, I make a vector, `any_char_vecs_pdbs`, with my list of PDB identities for the input for the function I'm about to create.

Next, I define a function, `analyze_proteins`. I use `function()` to define the function, and include `any_char_vec_pdb`s inside the parentheses to say that this function takes in the list of PDB identities as the input.

Inside the function, I then create `trimmed_list`, which will contain cleaned-up versions of the `any_char_vecs_pdbs` that include only some information for each PDB identity. I use `lapply()` to go through every PDB identity, load it with `read.pdb()`, and trim it with `trim.pdb()` to only include the specified chain and alpha carbons ("CA") for each original PDB identity.

Next, from the trimmed PDB list, I access the B-factor column for each PDB identity in the list. I use `lapply()` to go through every PDB identity in the trimmed PDB list. I use a new function(`pdb`)`pdbatomb`). The output is `beta_factors`, a list of numeric vectors which is the B-factor numbers for a given protein's alpha carbons, one beta_factor number per input PDB identity.

Then, I use `plotb3()` to create a plot of the B-factor number for the first protein in the `beta_factor` list. The `sse` argument adds information about secondary structure. `typ = "l"` tells it to plot a line graph. `main` and `ylab` label the plot.

Then, I have to plot the remaining PDB identities in the beta_factors list. The if(length(beta_factors)>1) essentially means if there are remaining identities beyond the first one, we're now dealing with them. The for(i in 2:length(beta_factors)) { makes a for loop, which says that we'll do the following for all the identities in the beta_factors list starting with the second one (since we already plotted the first one). The lines(beta_factors[[i]], col = i) adds the B-factor numbers for the identities (i) in the beta_factors list, one at a time as a new line on the plot, and col=i makes it so that there is a new color for each line added.

Finally, I run my new function analyze_proteins for my PDB list, any_char_vecs_pdbs, with analyze_proteins(any_char_vecs_pdbs).

To summarize, I create a function to extract and plot the B-factors from a list of PDBs, and to use it, I specify a list (character vector) of certain PDB character names to use for the function's input. The output is a line plot comparing the B-factors across this list of input PDBs.

*Citation: I used TritonGPT for help with troubleshooting/editing my code.