CODE EXPLANATION

* Installing Bio3d:

install.packages("bio3d", dependencies=TRUE)

install.packages("devtools")

library(devtools)

install\_bitbucket("Grantlab/bio3d", subdir = "ver\_devel/bio3d/")

library(bio3d)

* On your desktop (or some hard drive) create a folder where you will save the proteins that you download from the protein data bank
* The following code will direct R to that folder when it runs
* Place the file address in the green highlighted area; ensure that the slashes in your copied file address are forward slashes

setwd("H:/BIOCHEM/Total Proteins")

getwd()

* The following retrieves the pdb files from the created folder and then strips the filenames (when the files are saved, the file names aren’t in a format that may be used later in the code)

pdbfiles <- list.files(pattern="\*.pdb", full.names=TRUE) # Rerieving .pdb files

strippdb <- sapply(strsplit(pdbfiles, split='/', fixed=TRUE), function(x) (x[2]))

* This creates a vector (similar to a “list” type) of all of the filenames

pdbfilesnew = vector()

for (i in strippdb)

{

i = unlist(strsplit(i, split='.', fixed=TRUE))[1]

pdbfilesnew <- c(pdbfilesnew, i)

}

* Initializes a vector for the torsional angles
* Initializes 3 vectors to place the chi1, chi2, and chi3 values calculated for each protein

tor.prot = vector()

chi1 = vector()

chi2 = vector()

chi3 = vector()

x <- 1

* For every protein in the list created from the downloaded protein files

for (i in pdbfilesnew)

{

* Trim the protein file down, so every row of information (torsional angle) pertains to a single residue/amino acid
* The “hashtaged” code is commented out, so it doesn’t actually affect the script; however, if you have a protein that is causing errors, you may replace the non-hashtaged code with the hashtaged code below it; this allows you to type in the particular protein (the name of the protein from the PDB) for which you want to resolve the error and run the script for that particular protein as opposed to every protein in the list; ensure that the protein you type in has been downloaded and is in your chosen folder; see the next 4 lines below as an example of hashtaged vs non-hashtaged code

pdb.prot <- trim.pdb(read.pdb(i), "protein")

tor <- torsion.pdb(trim.pdb(read.pdb(i), "protein"))

#pdb.prot <- trim.pdb(read.pdb("4xo9"), "protein")

#tor <- torsion.pdb(trim.pdb(read.pdb("4xo9"), "protein"))

* This is no longer effectively used in the code, as a new method is used to generate torsional angles
* Formerly this created a matrix/table of all the torsional angles for a residue, each row is a single residue (by alpha carbons in the protein), this then labeled the columns by “Chain” + “-” + “residue number”; this allows you to search specific rows (specific amino acids) and find the psi/chi/etc. values

try(my\_residue\_labels <- paste(pdb.prot$atom$chain[pdb.prot$calpha], pdb.prot$atom$resno[pdb.prot$calpha], sep="-"))

while(nrow(tor$tbl) > length(my\_residue\_labels))

{

numover <- nrow(tor$tbl) - length(my\_residue\_labels)

numchange <- my\_residue\_labels[length(my\_residue\_labels)]

sub <- gsub(toString(length(my\_residue\_labels)), toString(length(my\_residue\_labels)+1), numchange)

my\_residue\_labels <- c(my\_residue\_labels, sub)

}

try(rownames(tor$tbl) <- my\_residue\_labels)

* Extracts data (of the type character/string) that contains information about the exact number and placement (chain/residue number) of an SSBOND and places it in variable ssbondindex
* Ex:



#raw = readLines("4xo9.pdb")

raw = readLines(strippdb[x])

inds = grep("SSBOND", raw)

ssbondindex <- raw[inds]

* This goes through every lines of ssbondindex and then every position in said line and puts information about the SSBOND into variables
  + 4 variables are made, 2 representing the chain and 2 representing the residue number
  + For example: 
  + Here, a variable a would be set as “A”; a variable b would be set as “42”; a variable c would be set as “A”; and a variable d would be set as “58”
  + The information extracted is the chain and residue numbers of the 2 parts of the protein that form the SSBOND, ex. Chain A residue 42 and chain A residue 58

for (i in ssbondindex)

{

s <- sapply(strsplit(i, split=' ', fixed=TRUE), `[`, 1)

if (s == "SSBOND")

{

a <- sapply(strsplit(i, split=' ', fixed=TRUE), `[`, 6)

b <- sapply(strsplit(i, split=' ', fixed=TRUE), `[`, 7)

v<-1

while(b == "")

{

b <- sapply(strsplit(i, split=' ', fixed=TRUE), `[`, 7+v)

v<-v+1

}

c <- sapply(strsplit(i, split=' ', fixed=TRUE), `[`, 7+v)

while(c == "")

{

c <- sapply(strsplit(i, split=' ', fixed=TRUE), `[`, 7+v)

v<-v+1

}

while(c == "CYS")

{

c <- sapply(strsplit(i, split=' ', fixed=TRUE), `[`, 7+v)

v<-v+1

}

d <- sapply(strsplit(i, split=' ', fixed=TRUE), `[`, 7+v)

while(d == "")

{

d <- sapply(strsplit(i, split=' ', fixed=TRUE), `[`, 7+v)

v<-v+1

}

string1 <- c(a, b)

string2 <- c(c, d)

* Calculates chi values
* Starts by selecting certain atoms (ex. The beta carbon, “CB”) from a particular chain and a particular residue; the code uses the variables created previously to do this
  + Here, sele1 is a list containing only the beta carbon, sele2 is a list containing the first sulfur group for the chi3 calculation, etc.
  + \*\*\*later in the code, for example when calculating the chi2, sele1 might contain more than one atom

#chi3chain1values

sele1 = atom.select(pdb.prot, elety=c("CB"), resno=as.numeric(string1[2]), chain=string1[1])

sele2 = atom.select(pdb.prot, elety=c("SG"), resno=as.numeric(string1[2]), chain=string1[1])

sele3 = atom.select(pdb.prot, elety=c("SG"), resno=as.numeric(string2[2]), chain=string2[1])

sele4 = atom.select(pdb.prot, elety=c("CB"), resno=as.numeric(string2[2]), chain=string2[1])

* Once it selects the 4 atoms needed for a given chi torsional angle, it makes a list of the indices (x,y,z) for each atom and puts this information into vector inds.xyz

inds.xyz = c(sele1$xyz, sele2$xyz, sele3$xyz, sele4$xyz)

* This takes the vector of indices and calculates the torsional angles between the atoms

test1<- try(torsion.xyz(pdb.prot$xyz[, inds.xyz]))

chi3 <- c(chi3,torsion.xyz(pdb.prot$xyz[, inds.xyz]))

#chi3chain2values

#sele1 = atom.select(pdb.prot, elety=c("CB", "SG"), resno=as.numeric(string2[2]), chain=string2[1])

#sele2 = atom.select(pdb.prot, elety=c("SG","CB"), resno=as.numeric(string1[2]), chain=string1[1])

#inds.xyz = c(sele1$xyz, sele2$xyz)

#test2<-try(torsion.xyz(pdb.prot$xyz[, inds.xyz]))

#chi23 <- c(chi3,torsion.xyz(pdb.prot$xyz[, inds.xyz]))

#chi2chain1values

sele1 = atom.select(pdb.prot, elety=c("CA", "CB", "SG"), resno=as.numeric(string1[2]), chain=string1[1])

sele2 = atom.select(pdb.prot, elety=c("SG"), resno=as.numeric(string2[2]), chain=string2[1])

inds.xyz = c(sele1$xyz, sele2$xyz)

try(torsion.xyz(pdb.prot$xyz[, inds.xyz]))

chi2 <- c(chi2,torsion.xyz(pdb.prot$xyz[, inds.xyz]))

#chi2chain2values

sele1 = atom.select(pdb.prot, elety=c("CA", "CB", "SG"), resno=as.numeric(string2[2]), chain=string2[1])

sele2 = atom.select(pdb.prot, elety=c("SG"), resno=as.numeric(string1[2]), chain=string1[1])

inds.xyz = c(sele1$xyz, sele2$xyz)

try(torsion.xyz(pdb.prot$xyz[, inds.xyz]))

chi2 <- c(chi2,torsion.xyz(pdb.prot$xyz[, inds.xyz]))

#chi1chain1values

sele1 = atom.select(pdb.prot, elety=c("N", "CA", "CB", "SG"), resno=as.numeric(string1[2]), chain=string1[1])

inds.xyz = c(sele1$xyz)

try(torsion.xyz(pdb.prot$xyz[, inds.xyz]))

chi1 <- c(chi1,torsion.xyz(pdb.prot$xyz[, inds.xyz]))

#chi1chain2values

sele1 = atom.select(pdb.prot, elety=c("N", "CA", "CB", "SG"), resno=as.numeric(string2[2]), chain=string2[1])

inds.xyz = c(sele1$xyz)

try(torsion.xyz(pdb.prot$xyz[, inds.xyz]))

chi1 <- c(chi1,torsion.xyz(pdb.prot$xyz[, inds.xyz]))

}

}

x <- x + 1

}

* This creates 3 excel files, one per chi vector created, thus all the angles placed into the chi1 vector go into one excel file, the second excel files contains the chi2 vector data, etc.
* The 3 excel files can be found in the folder that you originally set R to access to find the proteins

y <- as.character(chi1)

write.csv(y, file = 'chi1.csv', row.names = F)

y <- as.character(chi2)

write.csv(y, file = 'chi2.csv', row.names = F)

y <- as.character(chi3)

write.csv(y, file = 'chi3.csv', row.names = F)