Analysis Guide

version 0.1

Last generated: October 07, 2021

Table of Contents

Applying Difference Data to Structures in Chimera

Using Gnuplot

Introduction

The following is a guide to different types of analysis for molecular dynamics simulations. Relevant other references include the:

- AMBER Manual
- AMBER Mailing List
- VMD User's Guide
- Chimera User's Guide
- Gnuplot User's Guide

The scripts referenced in this guide can be found in this Github repository .

PDF Download

Cpptraj

Cpptraj is a useful data analysis tool for AMBER simulations. Detailed information about it can be found in the AMBER manual . *Cpptraj* can be used to strip the waters from the simulation and autoimage (center) the protein.

Since *cpptraj* is a program in AMBER, to run it, you would use a command like:

```
$ $AMBERHOME/bin/cpptraj -p WT_protein_system_wat.prmtop -i cpp
traj_strip.in
```

The `-p` flag specifies that a prmtop file is being read in, and the `-i` flag specifies your input file. Using *cpptraj* in this manner is known as using it in batch mode.

You can also run *cpptraj* without an input file. In that case, you would start with either

```
$ AMBERHOME/bin/cpptraj
cpptraj > parm WT_protein_system_wat.prmtop
```

or

```
$ $AMBERHOME/bin/cpptraj -p WT_protein_system_wat.prmtop
```

and follow through the list of commands line-by-line from your input file. Using *cpptraj* in this manner is helpful when trying to test analysis commands (because *cpptraj* is… finicky and the syntax can be easily messed up).

The following is an example input file which would strip and autoimage the first five trajectory files (with the `mdcrd` extension; this could also be `nc` depending on your scripted commands) and then write out a comprehensive NetCDF file with the information of the five trajectory files (i.e `cpptraj_strip.in`).

cpptraj_strip.in

```
reference WT_protein_system_wat_init0.rst
trajin WT_protein_system_wat_md1.mdcrd
trajin WT_protein_system_wat_md2.mdcrd
trajin WT_protein_system_wat_md3.mdcrd
trajin WT_protein_system_wat_md4.mdcrd
trajin WT_protein_system_wat_md5.mdcrd

autoimage
strip :WAT,K+

trajout WT_protein_system_1-5.nc cdf
```

It is important to note that if your system originally included waters that the strip command you use should be like `strip :WAT,K+ outprefix strip nobox`. The `outprefix strip nobox` part writes a new parmtop file that, following the current example naming scheme, would be `strip.WT_protein_system_wat.prmtop`. Without this part of the command, you'll need to use *parmed* (page 24) to strip the topology file, so that your imaged trajectory doesn't look like an angry sea urchin in VMD.

Now that a simulation has been processed, *cpptraj* can be used to gather a lot of different trajectory information, such as hydrogen bonds, normal modes, distance between residues, root mean square deviation (RMSD), and root mean square fluctuations (RMSF).

The command used to run *cpptraj* now should use a stripped prmtop file, which would have been the prmtop that you saved prior to solvation (or the newly created stripped prmtop).

```
$ AMBERHOME/bin/cpptraj -p WT_protein_system_vac.prmtop -i cppt
raj_analysis.in
```

An example incorporating several types of analysis is `cpptraj_analysis.in`.

cpptraj_analysis.in

```
trajin WT_protein_system_1-5.nc

autoimage

rms first out test_rms.dat :1-476 byres

## For correlation matrix
matrix out WT_protein_system_matrix_correl.dat name corr_mat \
 byres :1-476 correl

## For normal modes (evecs = eigenvectors)
matrix out WT_protein_system_covar_mat.dat name norm_mode :1-47
6@CA,P,C4',C2 covar
diagmatrix norm_mode out WT_protein_system_evecs.out vecs 100 r
educe \
 nmwiz nmwizvecs 100 nmwizfile WT_protein_system_100.nmd nmwizm
ask :1-476@CA,P,C4',C2

hbond out WT_protein_system_hbond.dat dist 3.0 avgout \
   WT_protein_system_hbond_avg.dat

rms first out WT_protein_system_total_bb_rms.dat :1-476@CA,P,O
3',O5',C3',C4',C5'

rmsd :1-476 first perres perresavg range 1-476 perresout \
   WT_protein_system_rmsd_byres.dat

atomicfluct :1-476 out WT_protein_system_rmsf_byres.dat byres

distance :476@PA :457@O3' out dist_PO_WT_protein_system.dat
```

The `:1-476` atom mask is specifying that these processes are run on all of the 476 system residues. `hbond` extracts out the time hydrogen binds are present, `rmsd` finds the RMSD (shocker!), `atomicfluct` finds the RMSF, and `dist` determines the distance between two specified atoms throughout the simulation. The `\` symbols signify a line continuation for those specific commands.

Plotting RMSD, RMSF, and Total Number of Hydrogen Bonds with gnuplot

Gnuplot is a freely available plotting utility that can be used to make publication-worthy images. The utility is command line operated, and can be used with a scripted input.

Gnuplot scripts end with the `.gnu` extension. To use gnuplot with a script, you would use a command like:

```
$ gnuplot scriptname.gnu
```

In the following script, `set encoding iso_8859_1` makes it so that the Å symbol can be encoded by using `{\305}`. The script also assumes that this is in a general directory, with additional system directories contained inside (think they're folder B, but you're in folder A), which is why the path to the data files is set as such. Another note to make is that the `$1/500` converts the frames to your nanosecond timescale–you'll need to change it based on what your simulation's conversion is. Additional details about gnuplot scripts can be found on the gnuplot documentation and in the gnuplot section (page 155) later in this guide.

`rmsd-etc.gnu`

(Thank's Alice!)

```
set encoding iso_8859_1
set term postscript enhanced color font "Arial,24";

set xlabel "Time (ns)"
set ylabel "RMSD ({\305})"
set key left bottom Left reverse
#set yrange [1.5:4]

set output "rmsds.eps";
plot "WT-system/WT_protein_system_total_bb_rms.dat" u ($1/50
0):($2) w lines s bezier t "Wild Type" lw 4, \
"MUT-A-system/MUT_A_system_total_bb_rms.dat" u ($1/500):($2) w
lines s bezier t "Mutation A" lw 4, \
"MUT-B-system/MUT_B_system_RNA_total_bb_rms.dat" u ($1/50
0):($2) w lines s bezier t "Mutation B" lw 4, \
"MUT-C-system/MUT_C_system_total_bb_rms.dat" u ($1/500):($2) w
lines s bezier t "Mutation C" lw 4;

set xlabel "Time (ns)"
set ylabel "Number of hydrogen bonds"

set output "hbonds.eps";
plot "WT-system/WT_protein_system_hbond.dat" u ($1/500):($2) w
lines s bezier t "Wild Type" lw 4, \
"MUT-A-system/MUT_A_system_hbond.dat" u ($1/500):($2) w lines
s bezier t "Mutation A" lw 4, \
"MUT-B-system/MUT_B_system_hbond.dat" u ($1/500):($2) w lines
s bezier t "Mutation B" lw 4, \
"MUT-C-system/MUT_C_system_hbond.dat" u ($1/500):($2) w lines
s bezier t "Mutation C" lw 4;

set xlabel "Residue number"
set ylabel "RMSF ({\305})"
set key top left Left reverse
set xrange [0:455]
#set yrange [0:7]

set output "rmsf.eps";
plot "WT-system/WT_protein_system_rmsf_byres.dat" u 1:2 w line
s t  "Wild Type" lw 4, \
"MUT-A-system/MUT_A_system_rmsf_byres.dat" u 1:2 w lines t  "Mu
tation A" lw 4, \
"MUT-B-system/MUT_B_system_rmsf_byres.dat" u 1:2 w lines t "Mut
ation B" lw 4, \
"MUT-C-system/MUT_C_system_rmsf_byres.dat" u 1:2 w lines t "Mut
```

```
ation C" lw 4;
```

Plotting Averages for RMSD (etc.) using R and gnuplot

Like in previous section (page 7), you can plot the averages of multiple simulations. You can also use gnuplot to shade in the standard deviation of these averages, but only once you've processed the data appropriately. Therefore, a script using the programming language R has been created to properly get averages and standard deviations for plotting with gnuplot. Once R has been installed on your computer, then you would run the script with

```
$ Rscript r-magic-rmsd-rmsf-hbond-5.r
```

If this is your first time installing R, however, then you'll also need to install the packages that the script needs. These have been commented out in the provided script, but you can also just do it before running the script by entering the R environment like:

```
$ R
> install.packages("data.table")
> install.packages("abind")
> quit()
Save workspace image? [y/n/c]: n
```

The averaging script has been built for 2 systems with 3 replicates, but sometimes you only need averages for one system, and sometimes you have more (or less!) than 3 replicates. So, the framework for up to 5 replicates has been created using comments, and you should modify the script accordingly. Also, it is important to use absolute paths for the script, so no tildes in place of `/username/home`!

rmagic-rmsd-rmsf-hbond-5.r

```
## Run this with "Rscript rmagic-rmsd-rmsf-hbond-5.r"
## (Assuming you've already installed R...)

#-----------------------------------------------#
#--Specify the paths to the Files from cpptraj--#
#-----------------------------------------------#

## This script has been pre-built for 2 systems with 3 replicat
es
## More or less than 3 reps (up to 5) can be achieved through
## Commenting or uncommenting

## Paths to the RMSD files
## Set A (system 1)
infile1A <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-1/WT_protein_system_total_bb_rms.dat")
infile2A <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-2/WT_protein_system_total_bb_rms.dat")
infile3A <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-3/WT_protein_system_total_bb_rms.dat")
#infile4A <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-4/WT_protein_system_total_bb_rms.dat")
#infile5A <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-5/WT_protein_system_total_bb_rms.dat")

##Set B (system 2)
infile1B <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-1/MUT_A_system_total_bb_rms.dat")
infile2B <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-2/MUT_A_system_total_bb_rms.dat")
infile3B <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-3/MUT_A_system_total_bb_rms.dat")
#infile4B <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-4/MUT_A_system_total_bb_rms.dat")
#infile5B <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-5/MUT_A_system_total_bb_rms.dat")

## Paths to the RMSF files
## Set A (system 1)
infile1AF <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-1/WT_protein_system_rmsf_byres.dat")
infile2AF <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-2/WT_protein_system_rmsf_byres.dat")
infile3AF <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-3/WT_protein_system_rmsf_byres.dat")
```

```
#infile4AF <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-4/WT_protein_system_rmsf_byres.dat")
#infile5AF <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-5/WT_protein_system_rmsf_byres.dat")

infile1BF <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-1/MUT_A_system_rmsf_byres.dat")
infile2BF <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-2/MUT_A_system_rmsf_byres.dat")
infile3BF <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-3/MUT_A_system_rmsf_byres.dat")
#infile4BF <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-4/MUT_A_system_rmsf_byres.dat")
#infile5BF <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-5/MUT_A_system_rmsf_byres.dat")

## Paths to the H-Bond files
## Set A (system 1)
infile1AH <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-1/WT_protein_system_hbond.dat")
infile2AH <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-2/WT_protein_system_hbond.dat")
infile3AH <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-3/WT_protein_system_hbond.dat")
#infile4AH <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-4/WT_protein_system_hbond.dat")
#infile5AH <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ WT-System-5/WT_protein_system_hbond.dat")

##Set B (system 2)
infile1BH <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-1/MUT_A_system_hbond.dat")
infile2BH <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-2/MUT_A_system_hbond.dat")
infile3BH <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-3/MUT_A_system_hbond.dat")
#infile4BH <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-4/MUT_A_system_hbond.dat")
#infile5BH <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/ MUT-A-System-5/MUT_A_system_hbond.dat")

#-----------------------------#
#--Define your outfile names--#
#-----------------------------#
```

```
## A is for infiles labeled A
## Each system gets an averaged file
## That is then used to plot with gnuplot
RMSDA <- "WT_protein_system_total_bb_rms_avgd.dat"
RMSDB <- "MUT_A_system_total_bb_rms_avgd.dat"

RMSFA <- "WT_protein_system_rmsf_byres_avgd.dat"
RMSFB <- "MUT_A_system_rmsf_byres_avgd.dat"

HBONDA <- "WT_protein_system_hbond_3trial_avgd.dat"
HBONDB <- "MUT_A_system_hbond_3trial_avgd.dat"

#-----------------------------------------------------------------
-----#
#-------Behind the Curtain: No Need to Modify Past This Lin
e-------#
#-----------------------------------------------------------------
-----#

## Use the data tables package to read in data frames
## Remove comment to install locally
#install.packages("data.table")
library(data.table)

## Use the abind package to combine data frames
## Remove comment to install locally
#install.packages("abind")
library(abind)

#------------------#
#--Begin with RMSD--#
#------------------#

## Reading each file as a data.table.
## Bonus - fread is much faster than read.csv
read1A <- fread(infile1A, header=TRUE)
read2A <- fread(infile2A, header=TRUE)
read3A <- fread(infile3A, header=TRUE)
#read4A <- fread(infile4A, header=TRUE)
#read5A <- fread(infile5A, header=TRUE)

read1B <- fread(infile1B, header=TRUE)
read2B <- fread(infile2B, header=TRUE)
read3B <- fread(infile3B, header=TRUE)
#read4B <- fread(infile4B, header=TRUE)
```

```
#read5B <- fread(infile5B, header=TRUE)

## Combine into one dataset
## Use both columns 1A and second column only for 2A and 3A
## Note: this makes it a matrix
combineA = abind(read1A, read2A[,2], read3A[,2], along=2)
#combineA = abind(read1A, read2A[,2], read3A[,2], read4A[,2], r
ead5A[,2], along=2)

combineB = abind(read1B, read2B[,2], read3B[,2], along=2)
#combineB = abind(read1B, read2B[,2], read3B[,2], read4B[,2], r
ead5B[,2], along=2)

## Change the column names so future life makes sense
## Your data are now Frame, infile1A RMSD, infile2A RMSD, infil
e3A RMSD
colnames(combineA) <- c("Frame", "RMSD1", "RMSD2", "RMSD3")
#colnames(combineA) <- c("Frame", "RMSD1", "RMSD2", "RMSD3", "R
MSD4", "RMSD5")

colnames(combineB) <- c("Frame", "RMSD1", "RMSD2", "RMSD3")
#colnames(combineB) <- c("Frame", "RMSD1", "RMSD2", "RMSD3", "R
MSD4", "RMSD5")

## Redefine as a data frame
combineA <- as.data.frame(combineA)

combineB <- as.data.frame(combineB)

## Append a column that's the average of RMSD cols
combineA$Average <- rowMeans(combineA[,2:4])
#combineA$Average <- rowMeans(combineA[,2:6])

combineB$Average <- rowMeans(combineB[,2:4])
#combineB$Average <- rowMeans(combineB[,2:6])

## Append a column that's the STDEV of RMSD cols
## The 1 means you're applying one function
## Which is the standard deviation function, sd
combineA$STDEV <- apply(combineA[,2:4], 1, sd)
#combineA$STDEV <- apply(combineA[,2:6], 1, sd)

combineB$STDEV <- apply(combineB[,2:4], 1, sd)
#combineB$STDEV <- apply(combineB[,2:6], 1, sd)
```

```
## Create a new variable with just Frame, Average, and STDEV
save_cols_A <- combineA[,c("Frame", "Average", "STDEV")]

save_cols_B <- combineB[,c("Frame", "Average", "STDEV")]

## Limit to 4 sig figs after decimal
save_cols_clean_A <- format(save_cols_A, digits=4)

save_cols_clean_B <- format(save_cols_B, digits=4)

#-------------------------------------------------------------------
-------#
#-------------------------RMSD OUTFILE
S-------------------------#
#-------------------------------------------------------------------
-------#

## Now write a tab-delimited outfile!
## Don't care about the index rownames because that's the frame
write.table(save_cols_clean_A, file = RMSDA, sep="\t", row.name
s=FALSE, quote=FALSE)

write.table(save_cols_clean_B, file = RMSDB, sep="\t", row.name
s=FALSE, quote=FALSE)

#----------------------#
#--CONTINUE WITH RMSF--#
#----------------------#

## Reading each file as a data.table.
## Bonus - fread is much faster than read.csv
read1AF <- fread(infile1AF, header=TRUE)
read2AF <- fread(infile2AF, header=TRUE)
read3AF <- fread(infile3AF, header=TRUE)
#read4AF <- fread(infile4AF, header=TRUE)
#read5AF <- fread(infile5AF, header=TRUE)

read1BF <- fread(infile1BF, header=TRUE)
read2BF <- fread(infile2BF, header=TRUE)
read3BF <- fread(infile3BF, header=TRUE)
#read4BF <- fread(infile4BF, header=TRUE)
#read5BF <- fread(infile5BF, header=TRUE)

## Combine into one dataset
## Use both columns 1A and second column only for 2A and 3A
```

```
## Note: this makes it a matrix
combineAF = abind(read1AF, read2AF[,2], read3AF[,2], along=2)
#combineAF = abind(read1AF, read2AF[,2], read3AF[,2], read4A
F[,2], read5AF[,2], along=2)

combineBF = abind(read1BF, read2BF[,2], read3BF[,2], along=2)
#combineBF = abind(read1BF, read2BF[,2], read3BF[,2], read4B
F[,2], read5BF[,2], along=2)

## Change the column names so future life makes sense
## Your data are now Residue, infile1A RMSF, infile2A RMSF, inf
ile3A RMSF
colnames(combineAF) <- c("Residue", "RMSF1", "RMSF2", "RMSF3")
#colnames(combineAF) <- c("Residue", "RMSF1", "RMSF2", "RMSF
3", "RMSF4", "RMSF5")

colnames(combineBF) <- c("Residue", "RMSF1", "RMSF2", "RMSF3")
#colnames(combineBF) <- c("Residue", "RMSF1", "RMSF2", "RMSF
3", "RMSF4", "RMSF5")

## Redefine as a data frame
combineAF <- as.data.frame(combineAF)

combineBF <- as.data.frame(combineBF)

## Append a column that's the average of RMSD cols
combineAF$Average <- rowMeans(combineAF[,2:4])
#combineAF$Average <- rowMeans(combineAF[,2:6])

combineBF$Average <- rowMeans(combineBF[,2:4])
#combineBF$Average <- rowMeans(combineBF[,2:6])

## Append a column that's the STDEV of RMSD cols
## The 1 means you're applying one function
## Which is the standard deviation function, sd
combineAF$STDEV <- apply(combineAF[,2:4], 1, sd)
#combineAF$STDEV <- apply(combineAF[,2:6], 1, sd)

combineBF$STDEV <- apply(combineBF[,2:4], 1, sd)
#combineBF$STDEV <- apply(combineBF[,2:6], 1, sd)

## Create a new variable with just Frame, Average, and STDEV
save_cols_AF <- combineAF[,c("Residue", "Average", "STDEV")]

save_cols_BF <- combineBF[,c("Residue", "Average", "STDEV")]
```

```
## Limit to 4 sig figs after decimal
save_cols_clean_AF <- format(save_cols_AF, digits=4)

save_cols_clean_BF <- format(save_cols_BF, digits=4)

#-----------------------------------------------------------------
--------#
#-------------------------RMSF OUTFILE
S------------------------#
#-----------------------------------------------------------------
--------#

## Now write a tab-delimited outfile!
## Don't care about the index rownames because that's the resid
ue number
write.table(save_cols_clean_AF, file = RMSFA, sep="\t", row.nam
es=FALSE, quote=FALSE)

write.table(save_cols_clean_BF, file = RMSFB, sep="\t", row.nam
es=FALSE, quote=FALSE)


#-----------------------#
#--CONTINUE WITH H-BOND--#
#-----------------------#

## Reading each file as a data.table.
## Bonus - fread is much faster than read.csv
read1AH <- fread(infile1AH, header=TRUE)
read2AH <- fread(infile2AH, header=TRUE)
read3AH <- fread(infile3AH, header=TRUE)
#read4AH <- fread(infile4AH, header=TRUE)
#read5AH <- fread(infile5AH, header=TRUE)

read1BH <- fread(infile1BH, header=TRUE)
read2BH <- fread(infile2BH, header=TRUE)
read3BH <- fread(infile3BH, header=TRUE)
#read4BH <- fread(infile4BH, header=TRUE)
#read5BH <- fread(infile5BH, header=TRUE)

## Combine into one dataset
## Use both columns 1A and second column only for 2A and 3A
## Note: this makes it a matrix
combineAH = abind(read1AH, read2AH[,2], read3AH[,2], along=2)
```

```
#combineAH = abind(read1AH, read2AH[,2], read3AH[,2], read4A
H[,2], read5AH[,2], along=2)

combineBH = abind(read1BH, read2BH[,2], read3BH[,2], along=2)
#combineBH = abind(read1BH, read2BH[,2], read3BH[,2], read4B
H[,2], read5BH[,2], along=2)

## Change the column names so future life makes sense
## Your data are now Frame, infile1A HB, infile2A HB, infile3A
HB
colnames(combineAH) <- c("Frame", "HB1", "HB2", "HB3")
#colnames(combineAH) <- c("Frame", "HB1", "HB2", "HB3", "HB4",
"HB5")

colnames(combineBH) <- c("Frame", "HB1", "HB2", "HB3")
#colnames(combineBH) <- c("Frame", "HB1", "HB2", "HB3", "HB4",
"HB5")

## Redefine as a data frame
combineAH <- as.data.frame(combineAH)

combineBH <- as.data.frame(combineBH)

## Append a column that's the average of RMSD cols
combineAH$Average <- rowMeans(combineAH[,2:4])
#combineAH$Average <- rowMeans(combineAH[,2:6])

combineBH$Average <- rowMeans(combineBH[,2:4])
#combineBH$Average <- rowMeans(combineBH[,2:6])

## Append a column that's the STDEV of RMSD cols
## The 1 means you're applying one function
## Which is the standard deviation function, sd
combineAH$STDEV <- apply(combineAH[,2:4], 1, sd)
#combineAH$STDEV <- apply(combineAH[,2:6], 1, sd)

combineBH$STDEV <- apply(combineBH[,2:4], 1, sd)
#combineBH$STDEV <- apply(combineBH[,2:6], 1, sd)

## Create a new variable with just Frame, Average, and STDEV
save_cols_AH <- combineAH[,c("Frame", "Average", "STDEV")]

save_cols_BH <- combineBH[,c("Frame", "Average", "STDEV")]

## Limit to 4 sig figs after decimal
```

```
save_cols_clean_AH <- format(save_cols_AH, digits=4)

save_cols_clean_BH <- format(save_cols_BH, digits=4)

#-----------------------------------------------------------------
---------#
#----------------------H-BOND OUTFILE
S-------------------------#
#-----------------------------------------------------------------
---------#

## Now write a tab-delimited outfile!
## Don't care about the index rownames because that's the frame

write.table(save_cols_clean_AH, file = HBONDA, sep="\t", row.na
mes=FALSE, quote=FALSE)

write.table(save_cols_clean_BH, file = HBONDB, sep="\t", row.na
mes=FALSE, quote=FALSE)
```

Now that that's over, we can once again plot it using gnuplot. This script also plots the standard deviation, which is why there are three parts of information being processed. It looks like you're plotting the same thing 3 times per graph. You are, in a way. Because gnuplot does key naming weirdly, you have to first plot the line you care about with the title for the key. Then you plot the standard deviation curves. That's dataset one. Then you plot dataset two. Finally, you plot those original curves again, because otherwise they're buried by the standard deviation plots. It's not counterintuitive, but it looks decent.

The reason behind this craziness is that gnuplot literally just stacks each plotted line over the others without any thought about what's being buried–whatever was plot last is on top. Think of it like a sandwich–if you put the peanut butter on the bread, you can't *really* see the full slice of bread anymore. You know that it is there, because you see the outline, but the entire piece of bread is obscured by the peanut butter. This gets worse as you add the jelly and the other slice of bread. The second piece of bread hides any pattern in the peanut butter or jelly. Now you probably get why you have to trick gnuplot.

avg-rmsd-etc.gnu

```
set encoding iso_8859_1
set term postscript enhanced color font "Arial,24";

set xlabel "Time (ns)"
set ylabel "RMSD ({\305})"
set key right bottom Left reverse maxrows 2
#set yrange [1.5:4]

# Other Berendsen inputs are $1/500, these Langevin are $1/100
set output "rmsds-avg.eps";
plot "WT_protein_system_total_bb_rms_avgd.dat" u ($1/100):($2)
w lines s bezier t "Wild Type" ls 1 lw 4, \
"WT_protein_system_total_bb_rms_avgd.dat" u ($1/10
0):($2+$3):($2-$3) w filledcurve fs solid 0.15 t "Std. Dev." l
s 1, \
"MUT_A_system_total_bb_rms_avgd.dat" u ($1/100):($2) w lines s
bezier t "Mutation A" ls 2 lw 4, \
"MUT_A_system_total_bb_rms_avgd.dat" u ($1/10
0):($2+$3):($2-$3) w filledcurve fs solid 0.15 t "Std. Dev." l
s 2, \
"WT_protein_system_total_bb_rms_avgd.dat" u ($1/100):($2) w lin
es s bezier notitle ls 1 lw 4, \
"MUT_A_system_total_bb_rms_avgd.dat" u ($1/100):($2) w lines s
bezier notitle ls 2 lw 4;

set xlabel "Time (ns)"
set ylabel "Number of hydrogen bonds"
#set key left bottom Left reverse

# Other Berendsen inputs are $1/500, these Langevin are $1/100
set output "hbonds-avg.eps";
plot "WT_protein_system_hbond_3trial_avgd.dat" u ($1/100):($2)
w lines s bezier t "Wild Type" ls 1 lw 4, \
"WT_protein_system_hbond_3trial_avgd.dat" u ($1/10
0):($2+$3):($2-$3) w filledcurve fs solid 0.15 t "Std. Dev." l
s 1, \
"MUT_A_system_hbond_3trial_avgd.dat" u ($1/100):($2) w lines s
bezier t "Mutation A" ls 2 lw 4, \
"WT_protein_system_hbond_3trial_avgd.dat" u ($1/10
0):($2+$3):($2-$3) w filledcurve fs solid 0.15 t "Std. Dev." l
s 2, \
"MUT_A_system_hbond_3trial_avgd.dat" u ($1/100):($2) w lines s
bezier notitle ls 1 lw 4, \
"MUT_A_system_hbond_3trial_avgd.dat" u ($1/100):($2) w lines s
bezier notitle ls 2 lw 4;
```

```
set xlabel "Residue number"
set ylabel "RMSF ({\305})"
set key left top Left reverse maxrows 2
#set yrange [0:7]
set xrange [0:455]

set output "rmsf-avg.eps";
plot "WT_protein_system_rmsf_byres_avgd.dat" u ($1):($2) w line
s t "Wild Type" ls 1 lw 4, \
"WT_protein_system_rmsf_byres_avgd.dat" u ($1):($2+$3):($2-$3)
w filledcurve fs solid 0.15 t "Std. Dev." ls 1, \
"MUT_A_system_hbond_3trial_avgd.dat" u ($1):($2) w lines t "Mut
ation A" ls 2 lw 4, \
"MUT_A_system_hbond_3trial_avgd.dat" u ($1):($2+$3):($2-$3) w f
illedcurve fs solid 0.15 t "Std. Dev." ls 2, \
"WT_protein_system_rmsf_byres_avgd.dat" u ($1):($2) w lines not
itle ls 1 lw 4, \
"MUT_A_system_hbond_3trial_avgd.dat" u ($1):($2) w lines notitl
e ls 2 lw 4;
```

parmed

If you've made the mistake of stripping all the water from a protein that had some crystal waters prior to solvation, you can generated a stripped topology file using *parmed*.

To do this follow:

```
$ $AMBERHOME/bin/parmed
> parm WT_protein_system_wat.prmtop
> strip :WAT,K+
> outparm strip.WT_protein_system_wat.prmtop
> quit
```

The bright side to having to use *parmed* is that you get to see some really adorable ASCII art.

There's plenty more that *parmed* can do, so check out the documentation (and, if you use Python, the Python package).

Hydrogen Bond Analysis (HBA)

cpptraj can be used to determine how long hydrogen bonds are present in a simulation using `hbond`.

Hydrogen bonds are a non-covalent structural force between a hydrogen atom and a nitrogen, oxygen, or fluorine atom. These hydrogen bonds arise because of the attraction between the hydrogen and the electronegative atom.

The data provided in the `WT_protein_system_hbond_avg.dat` data file contains the average length of simulation time that a hydrogen bond is present, defined through a distance of 3.0 Å.

Once those bonds have been singled out using *cpptraj*, then they can be further assessed through a program like Excel or LibreOffice Calc, or through scripting. Open the data file in a spreadsheet, ensuring that you will not be modifying the original `.dat` file. The columns pertinent to analysis are **#Acceptor** (the atom connected to the accepting hydrogen), **Donor** (the donating electronegative atom), and **Frac** (the fraction of the simulation that the bond is present for; 1 corresponds to 100%). After deleting the irrelevant columns (**DonorH**, **Frames**, and **AvgDist**), then you can sort the bonds by fraction, taking care to keep the donor and acceptor columns matched. Sorting by size allows for the removal of bonds that are unhelpful, which would be anything occurring less than 1% (0.001) of the simulation time. Delete those less than 0.001, and then sort the columns through

1. Alphabetically by **#Acceptor**
2. Alphabetically by **DonorH**
3. Numerically by **AvgDist**

All three options should be specified, because sometimes the hydrogen bond will appear, dissappear, and reappear, all of which are not intrinsically recognized through the averaging that *cpptraj* performs.

The actual analysis then has a few schools of thought. The first, is that different acceptors and donors of the same residue type are what you should be looking for, and thus you can take the sum of anything with those two specific residues in those respective positions. The other is that individual hydrogen bonds matter, and thus only things with the exact same name in the two columns should be added together. Overall, it depends on what goal is set for the analysis, or what argument that is trying to be made.

After the determination is made between by residue or by hydrogen, then you can begin comparison in whatever means you're going for. If you're comparing the bonding between complex A and complex B, then you would first sum all the

relevant bonds in the individual complexes, before matching line by line for specific residues/hydrogens. Once the two complexes are matched, then you can take the difference, and color or denote important changes in whatever manner makes sense in the spreadsheet. As an example, I performed something like that in the table below (page 26), where green shown matches within 30%, red show major changes of over 30%, purple corresponded to residues in one complex but not the other, and yellow corresponded to a residue that was changed between complexes (the SNP position).

Table: Example HBA Table

Complex A			Complex B		
#Acceptor	Donor	Frac	#Acceptor	Donor	Frac
ALA_269@O	PHE_273@N	0.1748	ALA_269@O	PHE_273@N	0.1645
ALA_269@O	LEU_313@N	0.1748			
ALA_356@O	SER_312@N	0.0324	ALA_356@O	SER_312@N	0.6705
ALA_356@O	SER_360@N	0.5218	ALA_356@O	SER_360@N	0.7717

HBA Averaging with R

As previously mentioned, the data that comes out of *cpptraj* can contain lines that have the same hydrogen bond for different lengths of time, and these need to be summed. This can be a tedious process, and also makes it difficult to average across simulations, since they may or may not have the exact same bonds. R is powerful enough to deal with this, and can thus be used to tidy up and average hydrogen bonding data for a simulation.

rmagic-hbond-avg-stringfix.r

```
## Run this with "Rscript rmagic-hbond-avg-stringfix.r"
## (Assuming you've already installed R...)

#-----------------------------------------------#
#--Specify the paths to the Files from cpptraj--#
#-----------------------------------------------#

## This script has been pre-built for a system with 3 replicate
s
## More or less than 3 reps (up to 5) can be achieved through
## Commenting or uncommenting

## Paths to the Hbond-avg files
## Set A (system 1)
infile1A <- Sys.glob("/absolute/path/to/the/analysis/files/for/
WT-System-1/WT_protein_system_hbond_avg.dat")
infile2A <- Sys.glob("/absolute/path/to/the/analysis/files/for/
WT-System-2/WT_protein_system_hbond_avg.dat")
infile3A <- Sys.glob("/absolute/path/to/the/analysis/files/for/
WT-System-3/WT_protein_system_hbond_avg.dat")
#infile4A <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/WT-System-4/WT_protein_system_hbond_avg.dat")
#infile5A <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/WT-System-5/WT_protein_system_hbond_avg.dat")

#-----------------------------#
#--Define your outfile names--#
#-----------------------------#

## A is for infiles labeled A
## Each system gets an averaged file
## That is then used with the H bond analysis script

A_avg <- "/absolute/path/to/the/averaged/file/WT-System_total_h
bond_avg_fs_double.dat"

## Explicitly set the number of the SNP/changed residue to matc
h
## This uses a regex to match with any characters before the _
## So ABC_100@O would become Changed_ResA@O
## Therefore you won't miss things at an atom-level by computin
g the residue
fix_stringA <- ".*_100"
fixed_stringA <-"Changed_ResA"
```

```
fix_stringB <- ".*_200"
fixed_stringB <-"Changed_ResB"

## How many data sets to add (use 4 after decimal)
sets <- 3.0000
#sets <- 5.0000

#-----------------------------------------------------------------
--------#
#---------Behind the Curtain: No Need to Modify Past This Lin
e---------#
#-----------------------------------------------------------------
--------#

## Use the data tables package to read in data frames
## Remove comment to install locally
#install.packages("data.table")
library(data.table)

## Use the tidyverse package to perform string replacement
## Remove comment to install locally
#install.packages("tidyverse")
library(tidyverse)

## Turn off scientific notation
options(scipen = 999)

#------------------------#
#--Read in Hbond Scripts--#
#------------------------#

## Reading each file as a data.table.
## Bonus - fread is much faster than read.csv
read1A <- fread(infile1A, header=TRUE)
read2A <- fread(infile2A, header=TRUE)
read3A <- fread(infile3A, header=TRUE)
#read4A <- fread(infile4A, header=TRUE)
#read5A <- fread(infile5A, header=TRUE)

colnames(read1A) <- c("Acceptor", "DonorH", "Donor", "Frames",
"Frac", "AvgDist", "AvgAng")
colnames(read2A) <- c("Acceptor", "DonorH", "Donor", "Frames",
"Frac", "AvgDist", "AvgAng")
colnames(read3A) <- c("Acceptor", "DonorH", "Donor", "Frames",
"Frac", "AvgDist", "AvgAng")
```

```
#colnames(read4A) <- c("Acceptor", "DonorH", "Donor", "Frame
s", "Frac", "AvgDist", "AvgAng")
#colnames(read5A) <- c("Acceptor", "DonorH", "Donor", "Frame
s", "Frac", "AvgDist", "AvgAng")

## Combine all the datasets into 1
bound <- rbind(read1A, read2A, read3A)
#bound <- rbind(read1A, read2A, read3A, read4A, read5A)

bound$Acceptor <- as.character(bound$Acceptor)
bound$DonorH <- as.character(bound$DonorH)
bound$Donor <- as.character(bound$Donor)
bound$Frac <- as.numeric(bound$Frac)
bound$Frames <- as.numeric(bound$Frames)
bound$AvgDist <- as.numeric(bound$AvgDist)
bound$AvgAng <- as.numeric(bound$AvgAng)

## Deal with the string
clean_boundA <- mutate_if(bound, is.character, str_replace_al
l, pattern=fix_stringA, replacement=fixed_stringA)

## Deal with the other string
clean_boundB <- mutate_if(clean_boundA, is.character, str_repla
ce_all, pattern=fix_stringB, replacement=fixed_stringB)

## Collapse repeat lines into themselves (i.e. add numbers toge
ther)
superbound <- aggregate(data=clean_boundB, cbind(Frames,Frac,Av
gDist,AvgAng)~Acceptor+Donor, FUN=sum)

## If DonorH matters, then use:
#superbound2 <- aggregate(data=bound, cbind(Frames,Frac,AvgDis
t,AvgAng)~Acceptor+DonorH+Donor, FUN=sum)

## Get average based on number of sets combined [This if for 3]
superbound$AvgFrame <- format(superbound$Frames / sets, digit
s=4, format="f")
superbound$AvgFrac <- format(superbound$Frac / sets, digits=4,
format="f")
superbound$AAvgDist <- format(superbound$AvgDist / sets, digit
s=4, format="f")
superbound$AAvgAng <- format(superbound$AvgAng / sets, digit
s=4, format="f")

save_cols_AH <- superbound[,c("Acceptor", "Donor", "AvgFrac")]
```

```
## Limit to 4 sig figs after decimal
save_cols_clean_AH <- format(save_cols_AH, digits=4)

colnames(save_cols_clean_AH) <- c("Acceptor", "Donor     ", "Av
gFrac")

## Now write a tab-delimited outfile!
## Don't care about the index rownames
#write.table(save_cols_clean_AH, file = A_avg, sep="\t", row.na
mes=FALSE, quote=FALSE)

## Write a pseudo-fixed width outfile
capture.output( print(save_cols_clean_AH, print.gap=3, row.name
s=FALSE), file = A_avg)
```

From here, you can use the HBA shell script. It won't have the listed issue of only being effective on single trials (since you just averaged them ?).

HBA: the Shell Script

As you can already tell, HBA is… tedious. So, it's been semi-scripted to identify what may be key residues between two systems. When using the following script as a stand-alone (and not using the previous R script (page 26) with it), here are a few points of caution:

- Situations where it'll work effectively:
    - Comparing single trials of 2 systems
    - Comparing the averages of 2 systems that have already been finagled together correctly.
- Situations where there's still some work that needs to be done:
    - Adapting it to get the average of 3 systems to be used in the comparison of 2 averaged systems

There was a lot of talk about averages in there. Averages are important, because they make your argument stronger. If one system had bond A for 90% of the time while B had it 2% of the time in trial one, but those are reversed in trial two, you can't actually say that bond is important without other structural information (like, per se, your entire active site being torn to shreds in trial 2, but it being perfectly intact in trial one). For now, the "smart" way to get the average would be to follow this set of things with Excel or Libre Office:

1. Open data A trial 1 in sheet alpha.

2. Copy data A trial 2 to sheet alpha, next to trial 1.
3. Repeat for remaining data A.
4. Insert/delete cells until each row of sheet alpha has matching information. Blank cells are ok! The words need to be the same.
5. Copy all of the names to one cell column in sheet alpha.
6. Compute the averages in a different cell column [ex: ($A3 + $A6 + $A9)/3] and copy that formula through for all.
7. Create a new set of columns that are equal to the named column and the average columns. [ex. In cell A12, do =A1; in cell A13, do =A10].
8. Copy those two columns into a text file and save. Now you have the averages for one data set!
9. Repeat all of those steps with data B.
10. Run the script with the averages of data A and B! Congratulations on HBA!

If you've used the R script (page 26), however, you don't need to worry about all of that! Hooray! And now, without further ado:

hbond-analysis.sh

```
#!/bin/bash

##########################################
##        Define your 2 datasets        ##
##      Your h-bond optional cutoff     ##
##         And your outfile name        ##
##########################################

## What two sets are you comparing?
infileA=TET_hedi_5mdC_hbond_avg_WTagain.dat
infileB=TET_5mC_RNA_hbond_avg.dat

## List cutoff as a percentage (20 is 20%)
cutoff=20

## Tag of differences for outfiles
filename=TET-5mdC-5mC-hbond-avg
setA=TET-hedi-5mdC
setB=TET-5mC-RNA

##########################################
##         Predefined variables         ##
##########################################

## You can change the file names, but
## it'll be annoying to change the variables

outfile1A=hbond-clean-1A.tmp
outfile2A=hbond-clean-2A.tmp
outfile3A=hbond-clean-3A.tmp

outfile1B=hbond-clean-1B.tmp
outfile2B=hbond-clean-2B.tmp
outfile3B=hbond-clean-3B.tmp

outfile4AB=hbond-clean-4AB.tmp
outfile5A=hbond-clean-5A.tmp
outfile5B=hbond-clean-5B.tmp
outfile6AB=hbond-clean-6AB.tmp
outfile7AB=hbond-clean-7AB.tmp
outfile8AB=hbond-clean-8AB.tmp
outfile9AB=hbond-clean-9AB.tmp
outfile10AB=$filename-abs-diff.dat

outfile11A=hbond-clean-11A.tmp
```

```
outfile11B=hbond-clean-11B.tmp
outfile12A=hbond-clean-12A.tmp
outfile12B=hbond-clean-12B.tmp

##########################################
##              Fileset A               ##
##  Make some files; do some analysis   ##
##########################################

## Clean the data. Remove lines less than
## 1% and print file with the
## 3 columns you want; keep header
## 1=Acceptor 3=Donor 5=Frac

awk 'NR == 1 {print $1,$3,$5}; NR > 1 { if ($5>0.0099) print
$1, $3, $5 }' $infileA > $outfile1A

## Sum duplicate acceptor/donor columns

awk 'NR == 1; {s1[$1,$2] = $1; s2[$1,$2] = $2; s3[$1,$2] +=
$3} END { for (i in s3) print s1[i], s2[i], s3[i]}' $outfile1A
> $outfile2A

## Clean up the output. Make alphabetical order
## by acceptor then by donor and print that
## in clean columns (with left-aligned AAs)

sort $outfile2A | awk '{ printf "%-15s %-15s %8s\n", $1, $2,
$3 }' > $outfile3A

##########################################
##              Fileset B               ##
##  Make some files; do some analysis   ##
##########################################

## Clean the data. Remove lines less than
## 1% and print file with the
## 3 columns you want; keep header
## 1=Acceptor 3=Donor 5=Frac

awk 'NR == 1 {print $1,$3,$5}; NR > 1 { if ($5>0.0099) print
$1, $3, $5 }' $infileB > $outfile1B

## Sum duplicate acceptor/donor columns
```

```
awk 'NR == 1; {s1[$1,$2] = $1; s2[$1,$2] = $2; s3[$1,$2] +=
$3} END { for (i in s3) print s1[i], s2[i], s3[i]}' $outfile1B
> $outfile2B

## Clean up the output. Make alphabetical order
## by acceptor then by donor and print that
## in clean columns (with left-aligned AAs)

sort $outfile2B | awk '{ printf "%-15s %-15s %8s\n", $1, $2,
$3 }' > $outfile3B

#########################################
##           Comparison Time           ##
##      How different are A and B?     ##
#########################################

## Get a list of things with matching Acceptor
## and Donor columns between the files

awk 'FNR==NR{a[$1,$2];next} (($1,$2) in a)' $outfile3A $outfile
3B | awk '{printf "%-15s %-15s\n", $1, $2}' > $outfile4AB

## Get rows with the matches from A & B

awk 'FNR==NR{a[$1,$2];next} (($1,$2) in a)' $outfile4AB $outfil
e3A > $outfile5A
awk 'FNR==NR{a[$1,$2];next} (($1,$2) in a)' $outfile4AB $outfil
e3B > $outfile5B

## Print a single file for comparison
## 1=Acceptor 2=Donor 3=A_Frac 6=B_Frac

paste $outfile5A $outfile5B > $outfile6AB

## Get the difference and print it

awk 'NR == 1 { print $1 "\t" $2 "\t\t" "Diff_A-B" }; NR>1, NF
> 0 { print $1 "\t" $2 "\t" ($6 - $3) }' $outfile6AB > $outfile
7AB
(head -n 1 $outfile7AB && tail -n +3 $outfile7AB | cat ) > $fil
ename-diff.dat

## Give it as absolute difference

awk '{ printf "%-15s %-15s %8s\n", $1, $2, ($3 >=0) ? $3 : 0 -
```

```
$3}' $outfile7AB > $outfile8AB

## Sort the file,
## keeping the header

(head -n 1 $outfile8AB && tail -n +3 $outfile8AB | sort -Rk 3n
r ) > $outfile9AB

## Show the difference as a percentage,
## keeping the header

awk 'NR == 1 { print $1 "\t" $2 "\t\t" "AbsDiff%" }; NR>1, NF
> 0 { printf "%-15s %-15s %8s\n", $1, $2, $3*100 }' $outfile9A
B > $outfile10AB

## Get the cutoff file

awk -v f=$cutoff 'NR == 1; NR > 1 {if ($3> f ) print}' $outfile
10AB > $filename-$cutoff-percent-cut.dat

##########################################
##             No Match List            ##
##  No match between A&B?  No problem.  ##
##########################################

## Get full list of no matches
awk 'NR==FNR{a[$1,$2];next} !(($1,$2) in a)' $outfile4AB $outfi
le3A > $outfile11A
awk 'NR==FNR{a[$1,$2];next} !(($1,$2) in a)' $outfile4AB $outfi
le3B > $outfile11B


## Print the no matches larger than cutoff
## giving it a header

awk -v f=$cutoff 'NR == 1; NR > 1 {if ( (100*$3) > f ) print}'
$outfile11A > $outfile12A
awk -v f=$cutoff 'NR == 1; NR > 1 {if ( (100*$3) > f ) print}'
$outfile11B > $outfile12B

## Clean the output and give the files

awk 'NR == 1 { print "#Acceptor" "\t" "Donor" "\t\t" "AbsDif
f%" }; NR > 1, NF > 0 { printf "%-15s %-15s %8s\n", $1, $2,
$3*100 }' $outfile12A > $setA-$cutoff-percent-cut-nomatch.dat
```

```
awk 'NR == 1 { print "#Acceptor" "\t" "Donor" "\t\t" "AbsDif
f%" }; NR > 1, NF > 0 { printf "%-15s %-15s %8s\n", $1, $2,
$3*100 }' $outfile12B > $setB-$cutoff-percent-cut-nomatch.dat


## Remove the temporary data files
rm hbond-clean*.tmp
```

In using this script, you set a cutoff of difference between A and B. Like before, 20% is considered significant, with bigger differences being important. You'll get three important outfiles–the information with anything exceeding the cutoff, and files with bonds that aren't matched between the two systems.

Correlational Analysis

Correlational analyses are computed through the example *cpptraj* script through the following lines:

```
## For correlation matrix
matrix out WT_protein_system_matrix_correl.dat name corr_mat \
 byres :1-476 correl
```

The first `matrix` line calculates the correlation matrix for the system and outputs that as a data file.

The second `matrix` line builds a covariance matrix for the system.

The `diagmatrix` line will calculate eigenmodes from quasiharmonic analysis using the generated covariance matrix. In normal human speak, that means the natural vibration of the system is computed through the application of fancy physics based on thermodynamics. One hundred vectors were specified to be calculated.

Plotting Correlation Matrices with Python

First things first: This will make the axes wrong if you try to insert them, which is why they're manually added to images following this script.

This script, like most scripts, should be modified based on the correlations that you are interested in. As it is now, it assumes that this script is in a directory that contains different directories with the data, and that you're looking for a specific residue of interest (here, it's 436). Python starts counting at 0, so residue 436 shows up as residue 435 in the script. Additionally, the axes need to be set explicitly. This is currently setup for a system with 455 residues. Thanks to Alice for `matrcorr_graph.py`.

matrcorr_graph.py

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import statsmodels.api as sm
from tables import *
from matplotlib.colors import LinearSegmentedColormap

data1 = np.genfromtxt("WT-system/WT_protein_system_matrix_corre
l.dat",delimiter=None)
data2 = np.genfromtxt("MUT-A-system/MUT-A-system_matrix_corre
l.dat",delimiter=None)
data3 = np.genfromtxt("MUT-B-system/MUT-B-system_matrix_corre
l.dat",delimiter=None)
data4 = np.genfromtxt("MUT-C-system/MUT-C-system_matrix_corre
l.dat",delimiter=None)

## Saving Data
data12 = np.subtract(data1,data2)
data21 = np.subtract(data2,data1)
np.savetxt('WT_minus_MUT-A_436.txt',data12[435],fmt='%1.2f')

data13 = np.subtract(data1,data3)
data31 = np.subtract(data3,data1)
np.savetxt('MUT-A_minus_MUT-B_436.txt',data13[435],fmt='%1.2f')

data14 = np.subtract(data1,data4)
data41 = np.subtract(data4,data1)
np.savetxt('WT_minus_MUT-C_436.txt',data14[435],fmt='%1.2f')

data23 = np.subtract(data2,data3)
data32 = np.subtract(data3,data2)
np.savetxt('MUT-A_minus_MUT-B_436.txt',data23[435],fmt='%1.2f')

data24 = np.subtract(data2,data4)
data42 = np.subtract(data4,data2)
np.savetxt('MUT-A_minus_MUT-D_436.txt',data24[435],fmt='%1.2f')

data34 = np.subtract(data3,data4)
data43 = np.subtract(data4,data3)
np.savetxt('MUT-B_minus_MUT-C_436.txt',data34[435],fmt='%1.2f')

## Self Plots

#Explicitly choose where to put x and y ticks
placesx = [0, 100, 200, 300, 400, 455]
```

```
placesy = [0, 55, 155, 255, 355, 455]
## Note: we're not using the inverted y-axis
## so therefore, this starts at bottom left

# Define those very x and y tick labels
labelsx = [0, 100, 200, 300, 400, 455]
labelsy = [455, 400, 300, 200, 100, 0]

sm.graphics.plot_corr(data1,normcolor=(-1.0,1.0),cmap='RdYlBu')
ax = plt.gca()
ax.axes.get_xaxis()
ax.set_yticks(placesx)
ax.set_xticklabels(labelsx, fontdict=None, minor=False)
ax.axes.get_yaxis()
ax.set_yticks(placesy)
ax.set_yticklabels(labelsy, fontdict=None, minor=False)
plt.savefig('WT_protein_system_mc.png')
plt.close('WT_protein_system_mc.png')

sm.graphics.plot_corr(data2,normcolor=(-1.0,1.0),cmap='RdYlBu')
ax = plt.gca()
ax.axes.get_xaxis()
ax.set_yticks(placesx)
ax.set_xticklabels(labelsx, fontdict=None, minor=False)
ax.axes.get_yaxis()
ax.set_yticks(placesy)
ax.set_yticklabels(labelsy, fontdict=None, minor=False)
plt.savefig('MUT-A-system_mc.png')
plt.close('MUT-A-system_mc.png')

sm.graphics.plot_corr(data3,normcolor=(-1.0,1.0),cmap='RdYlBu')
ax = plt.gca()
ax.axes.get_xaxis()
ax.set_yticks(placesx)
ax.set_xticklabels(labelsx, fontdict=None, minor=False)
ax.axes.get_yaxis()
ax.set_yticks(placesy)
ax.set_yticklabels(labelsy, fontdict=None, minor=False)
plt.savefig('MUT-B-system_mc.png')
plt.close('MUT-B-system_mc.png')

sm.graphics.plot_corr(data4,normcolor=(-1.0,1.0),cmap='RdYlBu')
ax = plt.gca()
ax.axes.get_xaxis()
ax.set_yticks(placesx)
```

```
ax.set_xticklabels(labelsx, fontdict=None, minor=False)
ax.axes.get_yaxis()
ax.set_yticks(placesy)
ax.set_yticklabels(labelsy, fontdict=None, minor=False)
plt.savefig('MUT-C-system_mc.png')
plt.close('MUT-C-system_mc.png')

## Actual Cross Plots

sm.graphics.plot_corr(data12,normcolor=(-1.0,1.0),cmap='RdYlB
u')
ax = plt.gca()
ax.axes.get_xaxis()
ax.set_yticks(placesx)
ax.set_xticklabels(labelsx, fontdict=None, minor=False)
ax.axes.get_yaxis()
ax.set_yticks(placesy)
ax.set_yticklabels(labelsy, fontdict=None, minor=False)
plt.savefig('WT_minus_MUT-A_436.png')
plt.close('WT_minus_MUT-A_436.png')

sm.graphics.plot_corr(data21,normcolor=(-1.0,1.0),cmap='RdYlB
u')
ax = plt.gca()
ax.axes.get_xaxis()
ax.set_yticks(placesx)
ax.set_xticklabels(labelsx, fontdict=None, minor=False)
ax.axes.get_yaxis()
ax.set_yticks(placesy)
ax.set_yticklabels(labelsy, fontdict=None, minor=False)
plt.savefig('MUT-A_minus_WT_436.png')
plt.close('MUT-A_minus_WT_436.png')

sm.graphics.plot_corr(data13,normcolor=(-1.0,1.0),cmap='RdYlB
u')
ax = plt.gca()
ax.axes.get_xaxis()
ax.set_yticks(placesx)
ax.set_xticklabels(labelsx, fontdict=None, minor=False)
ax.axes.get_yaxis()
ax.set_yticks(placesy)
ax.set_yticklabels(labelsy, fontdict=None, minor=False)
plt.savefig('WT_minus_MUT-B_436.png')
plt.close('WT_minus_MUT-B_436.png')
```

```
sm.graphics.plot_corr(data31,normcolor=(-1.0,1.0),cmap='RdYlB
u')
ax = plt.gca()
ax.axes.get_xaxis()
ax.set_yticks(placesx)
ax.set_xticklabels(labelsx, fontdict=None, minor=False)
ax.axes.get_yaxis()
ax.set_yticks(placesy)
ax.set_yticklabels(labelsy, fontdict=None, minor=False)
plt.savefig('MUT-B_minus_WT_436.png')
plt.close('MUT-B_minus_WT_436.png')

sm.graphics.plot_corr(data24,normcolor=(-1.0,1.0),cmap='RdYlB
u')
ax = plt.gca()
ax.axes.get_xaxis()
ax.set_yticks(placesx)
ax.set_xticklabels(labelsx, fontdict=None, minor=False)
ax.axes.get_yaxis()
ax.set_yticks(placesy)
ax.set_yticklabels(labelsy, fontdict=None, minor=False)
plt.savefig('MUT-A_minus_MUT-C_436.png')
plt.close('MUT-A_minus_MUT-C_436.png')

sm.graphics.plot_corr(data42,normcolor=(-1.0,1.0),cmap='RdYlB
u')
ax = plt.gca()
ax.axes.get_xaxis()
ax.set_yticks(placesx)
ax.set_xticklabels(labelsx, fontdict=None, minor=False)
ax.axes.get_yaxis()
ax.set_yticks(placesy)
ax.set_yticklabels(labelsy, fontdict=None, minor=False)
plt.savefig('MUT-C_minus_MUT-A_436.png')
plt.close('MUT-C_minus_MUT-A_436.png')

sm.graphics.plot_corr(data34,normcolor=(-1.0,1.0),cmap='RdYlB
u')
ax = plt.gca()
ax.axes.get_xaxis()
ax.set_yticks(placesx)
ax.set_xticklabels(labelsx, fontdict=None, minor=False)
ax.axes.get_yaxis()
ax.set_yticks(placesy)
ax.set_yticklabels(labelsy, fontdict=None, minor=False)
```

```
plt.savefig('MUT-B_minus_MUT-C_436.png')
plt.close('MUT-B_minus_MUT-C_436.png')

sm.graphics.plot_corr(data43,normcolor=(-1.0,1.0),cmap='RdYlB
u')
ax = plt.gca()
ax.axes.get_xaxis()
ax.set_yticks(placesx)
ax.set_xticklabels(labelsx, fontdict=None, minor=False)
ax.axes.get_yaxis()
ax.set_yticks(placesy)
ax.set_yticklabels(labelsy, fontdict=None, minor=False)
plt.savefig('MUT-C_minus_MUT-B_436.png')
plt.close('MUT-C_minus_MUT-B_436.png')
```

Modifications to Remove Axis Labels

The following modification can be used to have no axis labels.

```
sm.graphics.plot_corr(data43,normcolor=(-1.0,1.0),cmap='RdYlB
u')
ax = plt.gca()
ax.axes.get_xaxis().set_visible(False)
ax.axes.get_yaxis().set_visible(False)
plt.savefig('MUT-C_minus_MUT-B_436.png')
plt.close('MUT-C_minus_MUT-B_436.png')
```

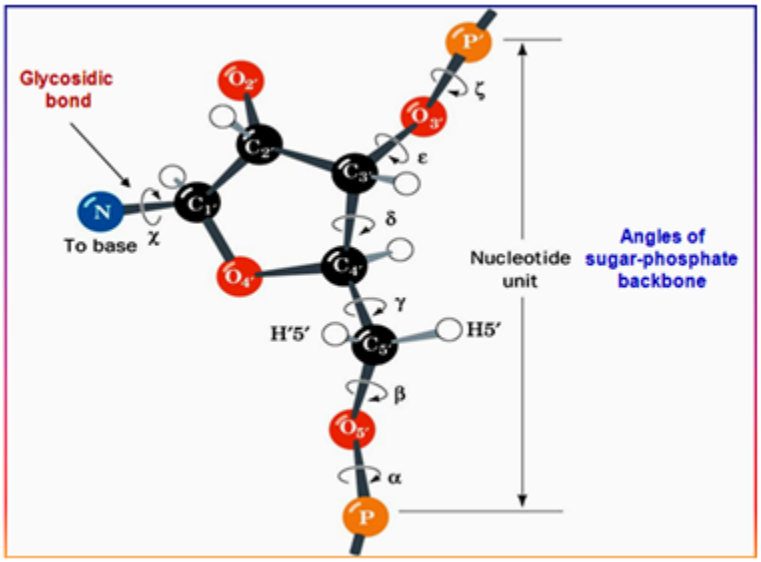
Modifications for (0,0) Origin

The following modification can be used to have a traditional (0,0) origin by inverting the y-axis.

```
#Explicitly choose where to put x and y ticks
placesx = [0, 100, 200, 300, 400, 455]
placesy = [0, 55, 155, 255, 355, 455]
## Note: we're using the inverted y-axis command
## so therefore, this starts at top left

# Define those very x and y tick labels
labelsx = [0, 100, 200, 300, 400, 455]
labelsy = [455, 400, 300, 200, 100, 0]

sm.graphics.plot_corr(data43,normcolor=(-1.0,1.0),cmap='RdYlB
u')
ax = plt.gca()
ax.axes.get_xaxis()
ax.set_xticks(placesx)
ax.set_xticklabels(labelsx, fontdict=None, minor=False)
ax.axes.get_yaxis()
ax.set_yticks(placesy)
ax.set_yticklabels(labelsy, fontdict=None, minor=False)
plt.gca().invert_yaxis()
plt.savefig('MUT-C_minus_MUT-B_436.png')
plt.close('MUT-C_minus_MUT-B_436.png')
```

Backbone Analysis

A lot of systems involve DNA or RNA. *Cpptraj* can be used to analyze the backbone structure in a variety of ways. Helpful lessons on nucleic backbone analysis be found on the Case Group page and on the website for the 3DNA program .



The different nucleic acid backbone angles, taken from NPTEL .

The different backbone angles can be seen in the figure above. These backbone angles have been defined through:

```
#Alpha=   :x-1@O3' :x@P :x@O5' :x@C5'
#Beta=    :x@P :x@O5' :x@C5' :x@C4'
#Gamma=   :x@O5' :x@C5' :x@C4' :x@C3'
#Delta=   :x@C5' :x@C4' :x@C3' :x@O3'
#Epsilon= :x@C4' :x@C3' :x@O3' :x+1@P
#Zeta=    :x@C3' :x@O3' :x+1@P :x+1@O5'

## Chi examples
## Pyrimidines (Y)= :x@O4' :x@C1' :x@N1 :x@C2     [C, T, U]
## Purines (R)= :x@O4' :x@C1' :x@N9 :x@C4         [A, G]
```

Thus, using the `dihedral` command in *cpptraj* can give you information on these angles. Specifying a dataset name (ex. `alpha449`) will allow multiple angles to be printed to the same out file.

```
#Residue 449 DG
dihedral alpha449 :448@O3' :449@P :449@O5' :449@C5' out RNA_bac
kbone-449-dihed.dat
dihedral beta449 :449@P :449@O5' :449@C5' :449@C4' out RNA_back
bone-449-dihed.dat
dihedral gamma449 :449@O5' :449@C5' :449@C4' :449@C3' out RNA_b
ackbone-449-dihed.dat
dihedral delta449 :449@C5' :449@C4' :449@C3' :449@O3' out RNA_b
ackbone-449-dihed.dat
dihedral epsilon449 :449@C4' :449@C3' :449@O3' :450@P out RNA_b
ackbone-449-dihed.dat
dihedral zeta449 :449@C3' :449@O3' :450@P :450@O5' out RNA_back
bone-449-dihed.dat
dihedral chi449 :449@O4' :449@C1' :449@N9 :449@C4 out RNA_backb
one-449-dihed.dat
```

Double-stranded nucleic acids can also be studied with respect to other base pairs. Information on different base pairs can be gathered by using the `nastruct` command in *cpptraj*. This command will automatically determine what is paired together, and nonstandard residues can be calculated based on the original base it was derived from (using `resmap`. The `nastruct` command has predetermined prefixes (`BP.`; `BPstep.`; and `Helix.`), but you specify the rest of the filename and extension after `naout`. An example is shown below.

```
nastruct master resrange 431,432,433,434,435,446,447,448,449,45
0 naout master.dat \
 resmap 5xC:C calcnohb
```

Because `nastruct` will automatically match pairs, the output will need to be cleaned up before plotting any data. This can be done pretty easily using awk.

```
$ awk 'NR == 1 || NR % 5 == 2' BP.master.dat > BP-A2-T2.dat
$ awk 'NR == 1 || NR % 4 == 2' BPStep.master.dat > BPstep-A2-C
3.dat
$ awk 'NR == 1 || NR % 4 == 2' Helix.master.dat > Helix-A2-C3.d
at
```

The `NR == 1` will print the first row (the header) and the `NR % 4 == 2` will print every 4th row starting with the second row.

Using 3DNA

Another program that can be used to analyze DNA is 3DNA program . The program can be downloaded after making an account on the 3DNA forum . A downside to using 3DNA is that you cannot analyze your entire trajectory–only snapshots in the form of PDBs. That said, the information is printed very cleanly, and the analysis takes seconds.

> ☑ **Tip:** Use a clustering analysis (page 53) to help pick your snapshots.

After installation and selecting your snapshots, the following command will be used to gather information on form, sugar puckering, and more.

```
$ find_pair WT_protein_system_md50.pdb | analyze
```

3DNA also has 2 other programs in testing, SNAP and DSSR. Help for any 3DNA analysis can be acquired by doing the executable name with the `-h` flag.

Secondary Structure

Secondary structure analysis can identify changes in secondary structure across a simulation or differences between systems. There are several types of secondary structure, including alpha helices, beta sheets, turns, and loops.

`cpptraj` has a keyword for secondary structure– `secstruct`. If your system has 350 protein residues, the command to use would be:

```
secstruct :1-350 out WT_protein_system_secstruct.gnu
```

This creates a `gnuplot` file for plotting the secondary structure (color) for each residue (y-axis) across time (x-axis). The file that is generated by `cpptraj`, however, is not ideal and may crash `gnuplot`. That initial `gnuplot` file can be modified by using a second script, `secstruct-gnu-fix.sh`.

```
#!/bin/bash
## Written for GNU sed (MacOS needs any -i <> to be -i '' <>)

## Name designations, without file extensions
## You can use paths
gnu_files="sys1_secstruct.gnu
sys2_secstruct.gnu
sys3_secstruct.gnu"

out_pngs="sys1_secstruct.png
sys2_secstruct.png
sys3_secstruct.png"

## Loop through gnu_files and out_pngs, modifying the gnu_file
s appropriately
clean_gnuplot_files()
{
set $out_pngs
for gnu_file in $gnu_files; do

## Print out which fileset of all of them that you're on
echo Gnuplot: "$gnu_file" PNG: "$1"

## new_lines contains the updated header information for the gn
uplot files.
## Update `xrange`, `yrange`, and the `splot (\$1/100)`variable
s  with what
## you need for your system.
## This is set up for custom ytics, but you can toggle the comm
enting
## if you don't have an infuriatingly numbered system.
## This uses a muted color scheme from https://personal.sron.n
l/~pault/
## but you can change it (`set palette defined`).
## Use double quotes around everything so that variables get ev
aluated,
## with the caveat that all other double quotes must be escaped
## (looking at you, ytics).
## The $1 refers to $out_pngs being set in fun().
## You need \\\\ for the proper number of escapes to print a si
ngle \ with ex
new_lines="set terminal pngcairo size 2560,1920 font \"Helvetic
a,48\";
set size 0.96,1
set encoding iso_8859_1
```

```
set pm3d map corners2color c1
set xtics nomirror out
#set ytics nomirror
set ytics (\"100\" 0, \"150\" 50, \"200\" 100, \"250\" 150, \"3
00\" 200, \\\\
\"350\" 250, \"400\" 300, \"BR\" 325, \"\" 335, \"1000\" 350,
\"1050\" 400) \\\\
border nomirror out
set cbrange [  -0.500:   7.500]
set cbtics    0.000,7.000,1.0
set palette maxcolors 8
set palette defined (0 \"#DDDDDD\",1 \"#AA4499\",2 \"#88225
5\", 3 \"#CC6677\",\\\\
4 \"#DDCC77\",5 \"#999933\", 6 \"#117733\",7 \"#44AA99\")
set cbtics(\"None\"    0.000,\"Para\"    1.000,\"Anti\"    2.00
0,\"3-10\"    \\\\
3.000,\"Alpha\"    4.000,\"Pi\"    5.000,\"Turn\"    6.000,\"Be
nd\"    7.000)
set xlabel \"Time (ns)\"
set ylabel \"Residue\"
set yrange [   0.000: 430.000]
set xrange [   0.000: 200.000]
set output \"${1}\"
splot \"-\" u (\$1/100):2:3 with pm3d notitle"

## ex is a command-line version of vi -- the << eof tells it t
o wait until eof
## :1,13d deletes the first 13 lines (bad header)
## :%s line sets up gnuplot for scripts (pause -1 assumes inter
active gnuplot)
## 1 insert will insert before the 1st line (inserts everythin
g until . given)
## $new_lines is what gets inserted (the {%?} evaluates it)
ex ${gnu_file} << eof
:1,13d
:%s/pause -1/set output/g
1 insert
${new_lines%?}
.
:wq
eof

## Increment $out_pngs and exit the function
shift
done
```

```
}

## Set up a function to go through the gnuplot scripts
run_gnuplot()
{
  for gnu_file in $gnu_files; do
    ## Print a status report
    echo Processing ${gnu_file} now
    gnuplot ${gnu_file}
  done
}

## Run the function
clean_gnuplot_files

## Run the gnuplot scripts too, while we're at it
run_gnuplot
```

Clustering

A cluster analysis can be used to group like things. In terms of trajectories, that means that it can be used to group parts of simulations that share certain characteristics. Let's say that in your system, you're really interested in the distance between two specific helices. You can use clustering to categorize your entire simulation time into groups that are based off the distance between them (i.e., 2-3 Å, 3-4 Å, 4-5 Å, etc).

If you're looking to do a clustering analysis, it's probably because you're thinking about doing quantum mechanics/molecular mechanics (QM/MM). Thus, you'll want to pick your clustering criteria based on what you're trying to study with QM/MM. Similarly, you'll want to generate 10-20 clusters to be used as snapshots for the QM/MM optimization. These clusters should be based on the the *entire* simulation–meaning every replicate.

This example reads in three full replicate trajectories and then autoimages them. From there, the distance of interest is found and given a tagged label (`d1`). Similarly, the root mean square deviation information is found and given the tagged label `rm0`. You can choose to write out either of these by specifying the data file to save them too. Then, kmeans clustering is done. The first cluster is given a tag (`C0`), uses kmeans, writes 10 clusters using data `d1`, and then writes out a bunch of files. These files contain summary information, which frame most resembles the average, the option to write a representative structure as a PDB file, and then information on cluster population over time.

```
trajin /absolute/path/to/the/file/WT_protein_system_wat_image
d_1-100.nc
trajin /absolute/path/to/the/file/WT_protein_system_wat_image
d_1-100.nc
trajin /absolute/path/to/the/file/WT_protein_system_wat_image
d_1-100.nc

autoimage

## FE-methyl H
distance d1 :455 :436@H11

## rms info
#rms rm0 :1-455@CA out WT_protein_system_rms_fromclust.dat
rms rm0 :1-455@CA

## k-means based on FE-H11 distance
cluster C0 kmeans clusters 10 data d1 \
  info WT_protein_system_H11_clust_detail_info.dat \
  out WT_protein_system_H11_clustnum_v_time.dat \
  summary WT_protein_system_H11_clust_summary.dat \
  avgout WT_cluster_H11 avgfmt pdb cpopvtime WT_protein_syste
m_H11_popvtime.dat

## k-means based on FE_H11 and rms
cluster C1 kmeans clusters 10 data rm0,d1 \
 info WT_protein_system_H11_rms_clust_detail_info.dat \
 out WT_protein_system_H11_rms_clustnum_v_time.dat \
 summary WT_protein_system_H11_rms_clust_summary.dat \
 avgout WT_cluster_H11_rms avgfmt pdb \
 cpopvtime WT_protein_system_H11_rms_popvtime.dat

## k-means based on rms distances
cluster C2 kmeans clusters 10 data rm0 \
 info WT_protein_system_rms_clust_detail_info.dat \
 out WT_protein_system_rms_clustnum_v_time.dat \
 summary WT_protein_system_rms_clust_summary.dat \
 avgout WT_cluster_rms avgfmt pdb cpopvtime WT_protein_system_r
ms_popvtime.dat
```

This is by no means the only way to do clustering, and *cpptraj* has more options than just kmeans. This website has a really great explanation of different clustering methods if you'd like to read more.

Sometimes the PDB that is written out from clustering (if requested) doesn't look quite right. That's because the PDB is an average representation. Thus, it may be ideal to write out the PDB noted in the summary file as the centroid for the cluster. This can also be done with *cpptraj*.

```
autoimage

## Write out the specific PDBs identified with clustering
trajout WT_protein_system_c0_frame_10.pdb pdb onlyframes 10
trajout WT_protein_system_c1_frame_37.pdb pdb onlyframes 37
trajout WT_protein_system_c2_frame_748.pdb pdb onlyframes 748
trajout WT_protein_system_c3_frame_1234.pdb pdb onlyframes 1234
trajout WT_protein_system_c4_frame_5257.pdb pdb onlyframes 5257
trajout WT_protein_system_c5_frame_8924.pdb pdb onlyframes 8924
```

If you need to use the PDB made with *cpptraj* for QM/MM with TINKER, then you will need to recenter it. TINKER is written so that the center of mass is located at the origin; AMBER centers the mass in the periodic box. So, your *cpptraj* script would look like this:

```
autoimage

## TINKER uses origin as COM, by default cpptraj uses box cente
r
## For center command, chose a mask that you'd use for cpptraj
analysis
## (probably protein, ligands, metals, and DNA/RNA)
center :1-455 origin mass

## Write out the specific PDBs identified with clustering
trajout WT_protein_system_com_c0_frame_10.pdb pdb onlyframes 10
trajout WT_protein_system_com_c1_frame_37.pdb pdb onlyframes 37
trajout WT_protein_system_com_c2_frame_748.pdb pdb onlyframes 7
48
trajout WT_protein_system_com_c3_frame_1234.pdb pdb onlyframes
1234
trajout WT_protein_system_com_c4_frame_5257.pdb pdb onlyframes
5257
trajout WT_protein_system_com_c5_frame_8924.pdb pdb onlyframes
8924
```

Custom Settings at Start-Up

Visual Molecular Dynamics (VMD) is a very powerful program (available for Unix, MacOS X, and Windows) that allows users to visualize their MD data.

VMD allows users to create a `.vmdrc` file that is sourced the program's start-up. This file should be located in your home directory (`/home/username`).

My `.vmdrc` file looks like:

```
display projection orthographic
menu main on
display nearclip set 0
axes location off
color Display Background white
display depthcue off
atomselect macro noh {not (mass 1.008 and charge < 0.25)}
set env(VMDFILECHOOSER) FLTK
color Labels Atoms black
color Labels Bonds black
label textthickness 2.0
```

The explanation:

- The default projection is "perspective," which will make things closer to the screen larger. Instead of a square box in orthographic, perspective is more of a triangular box shape.
- Using `menu main on` ensures that you will have the main menu (which is important if you haven't memorized the command prompt for every single thing VMD does) at all times.
- Nearclip controls how distinctly and crisply items pop out of the screen; the most clean view is achieved at lower values.
- The axes are turned off because they are relative and any generated images or movies shouldn't include them.
- The default background color is black. Any images or movies that are generated should be made with a white background.
- Depthcueing is the front-to-back shading of loaded molecules. When `depthcue` is off, there is no shading/automatic transparency.
- The default of VMD removes all hydrogens. By using the `atomselect` line, any polar hydrogen atoms are shown.

- `FLTK` is an independent file chooser that should effectively be the default for most *nix systems. This line is especially important when using Mac OS, due to some issues with trying to use a Mac-based file chooser. The other choice for VMD is `TK`.
- The default atom labels are green. Black is way easier to see on the white background.
- The default bond labels are also green. Again, black on white is easier to read.
- The default labels are thin and hard to read. This makes them bolder.

Files from the Command Line

VMD has been designed to operate from the command line. This allows files to be loaded directly at start up. The commands to load a prmtop file and an inpcrd file that have just been generated would be:

```
$ vmd -parm7 name_of_file.prmtop -rst7 name_of_file.inpcrd
```

The `-parm7` flag signifies an AMBER7 prmtop and the `-rst7` flag signifies an AMBER7 restart file. Don't let the AMBER7 scare you–AMBER switched formats for prmtops and restarts several years ago, and the name stuck around to mean anything file generated with or after AMBER7. If you wanted to look at a trajectory that has not had `ioutfm` explicitly set to 0 in the mdin files when working with AMBER16 (or later), then you would use:

```
$ vmd -parm7 name_of_file_wat.prmtop -netcdf name_of_file_wat_m
d25.mdcrd
```

or for a restart file which has not had `ntxo` explicitly set to 1 in the mdin files

```
$ vmd -parm7 name_of_file_wat.prmtop -netcdf name_of_solvated_f
ile_wat_md.rst
```

The `rst` file extension is just the last saved trajectory, and the `mdcrd` file extension contains all of the trajectory information for that segment (whose time frame changes based on the input settings). NetCDF is a condensed standardized format, and is really what these files have been saved as because that is the default, starting with AMBER16. Previous editions of AMBER wrote the `mdcrd` and `rst` files in ASCII format, and the flag to read those in would be `-crd`. PDBs can be loaded with the the `-pdb` flag.

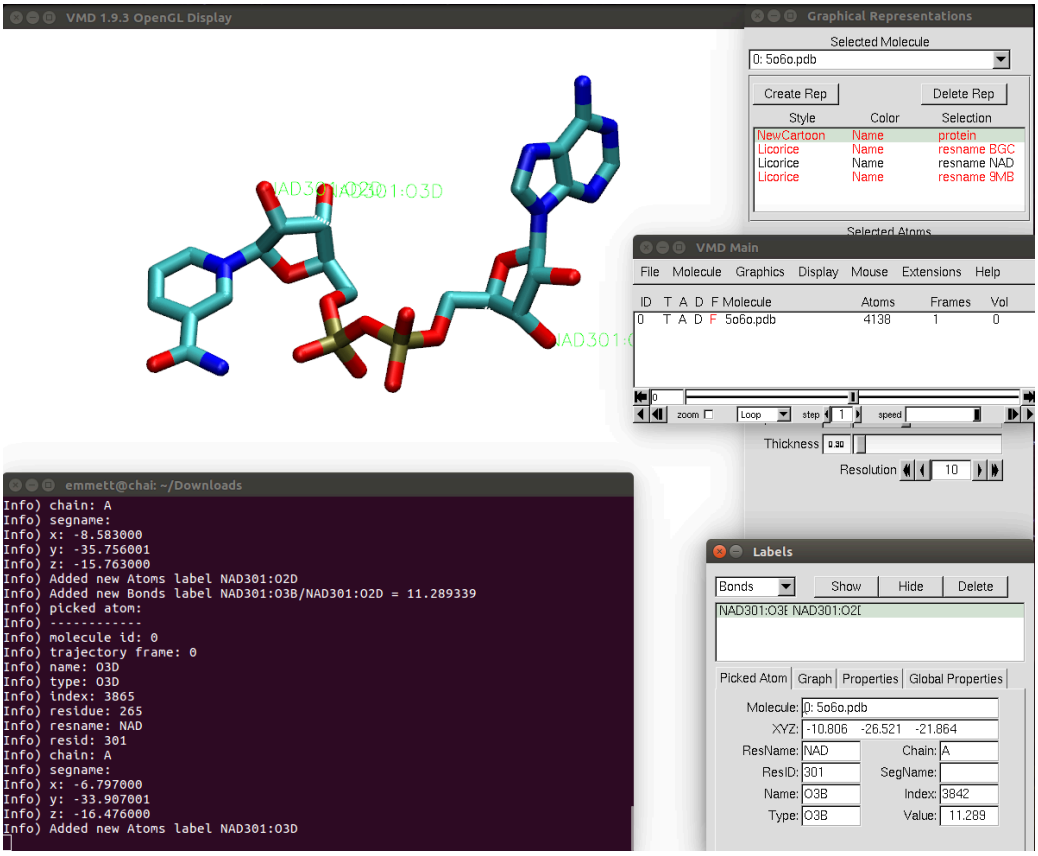
Changing Your System's Orientation

Have you ever thought to yourself “this is the most confusing thing I’ve ever thought about, and I wish I could move it around in 3D?” No? Well, apparently some researchers did at some point, because VMD allows you to rotate the system and zoom in on specific parts. They also enabled something called `hot keys`. Pressing `r` on the keyboard will either start or stop a system from rotating along the y-axis (get it, `r` for rotate?). Pressing `t` allows the system to be translated (get it, `t` for translate?). This means that you can move it around the screen (which is especially helpful if you’ve zoomed in by scrolling). Sometimes in VMD there are also some weird looking half in and out bonds (or times it just looks like your system is broken in that specific view). That can be fixed by pressing `t`, right-clicking with the mouse, and dragging downward.

There are some other hot keys that you can read about in the VMD User’s Guide, but these will at least get you to stop asking yourself why the heck your protein won’t stop spinning.

Labels

VMD allows you to label atoms, distances, angles, or dihedrals in several ways.



The Terminal, Display, and Labels GUI in VMD.

Atoms

One way to label atoms is by following `Mouse → Label → Atoms` in the main menu and then right-clicking on the atom/residue of interest. This will print some information to the Terminal that VMD is operating from, including what it was that was named, which can be helpful, since the default labeling color is neon green. Another way is to left click on the display window, hit `1` on the keyboard, and then click on the atom that you want information on using the mouse. Finally, following `Graphics → Labels` will pull up all the information printed to the Terminal in a GUI. These different labeling types are shown in the figure above.

Distances

Distances between two atoms can be labeled by left-clicking on the display window, hitting `2` on the keyboard. Your cursor then becomes a cross, which can be used to select the two specific atoms by left-clicking on one and then the other. Like with the atom information, the distance will be listed on the screen, printed to the Terminal, and shown in the labels GUI created through following `Graphics → Labels` (once you've changed `Atoms` to `Bonds` in the upper left corner).

Angles

Angles between three atoms can be labeled by left-clicking the display window, hitting `3` on the keyboard. Your cursor then becomes a cross, which can be used to select the three specific atoms by left-clicking on them in succession. The order you select them in makes a difference, though, so think through the angle you're interested in before making your selection. Like before, the angle will be listed on the screen, printed to the Terminal, and shown in the labels GUI created through following `Graphics → Labels` (once you've changed `Angles` in the upper left corner).

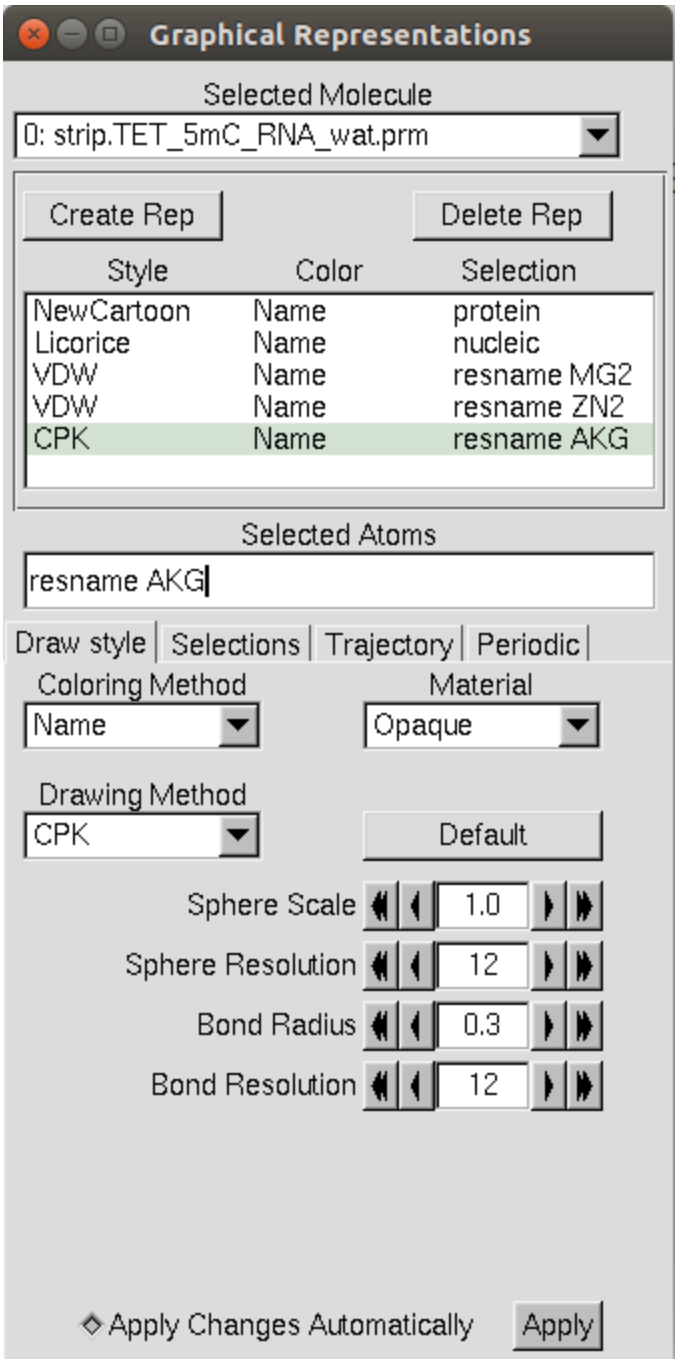
Dihedrals

Dihedrals between four atoms can be labeled the same was as angles (page 61), except instead of hitting `3`, you hit `4` on the keyboard and the atoms of interest. The atom order will also make a difference here, so be careful.

Graphical Representations

VMD has quite a few options for changing the view of your system. These are found by following `Graphics → Representations` in the VMD main menu. For any loaded system, the default (unless you've changed some settings) is to show everything using the `Lines` drawing method. Lines is helpful sometimes, but overwhelmingly, there are some common trends in how people create images for publication/presentations, and those are typically the representations that people use when visualizing their data all the time. To start, type `protein` into the `Selected Atoms` box and press enter. Then change the `Drawing Method` to `New Cartoon` and hit enter again. Now you have created your first representation for the protein. To add more layers, hit the `Create Rep` button. VMD will automatically generate a second representation of what you just made, and now you can edit that, hitting enter after every change.

Other common trends for VMD visualization are to show metal ions using the `VDW` drawing method (please… I'm begging you… call them "spheres" and NOT "van der Waal's balls"), using the `Licorice` drawing method for nucleic acids, and showing any cofactors with the `CPK` drawing method. However, you should really use what makes sense to you, until you're told to change it. The advice I have for you is to play around with these settings. Two quick other notes: 1) under `Materials`, there's a `Transparent` option, which while it may not look it onscreen, will actually look transparent in an image and 2) `Coloring Method` has a `Color ID` option, so that you can make entire portions of your protein one color. Finally, an example of a complete representation is shown below.



A complete graphical representation for a system.

VMD's Syntax

Ah, yes, you've just learned about `Selected Atoms`. Now's a good time to let you know VMD is a grammar-obsessed jerk who wants everything to be stated in exactly the right way. Here's a list of things that VMD will accept:

- `protein`: the protein
- `nucleic`: any nucleic acid residues
- `all not water`: everything in the structure that isn't water. Any keyword, like nucleic or protein, can be used here.
- `all not resname MG2`: everything in the structure that doesn't have the residue name MG2.
- `resname MG2`: anything in the PDB with the residue name of MG2. Any residue name (*cough* think of non-standard residues here *cough*) can be used with the `resname` command, provided it's found in the protein structure.
- `resid 244`: the residue corresponding to the number 244 in the PDB.
- `all within 5 of resname MG2`: everything in the structure within 5 Å residue name MG2.
- `resid 1 to 125`: all residue numbers from 1 through 125

To summarize: it's powerful, but if you mess up, VMD won't necessarily let you know that, and you'll just think you've lost a critical part of your structure (like a zinc in your active site that they entirety of everything you've ever cared about in research). When in doubt, save a PDB and check it using gedit or vi (page 0) for what you think is missing.

Saving/Loading Graphical Representations

If you're going to be making a lot of images, or just revisiting the same structure files over and over and over, you'll probably want to save visualization state. This essentially saves the information for the loaded compound, such as frames loaded in, and the graphical representation information. To do this, either follow `File → Save Visualization State` in the main menu, or type:

```
$ save_state name-of-saved-state.vmd
```

into the command line where VMD is operating from.

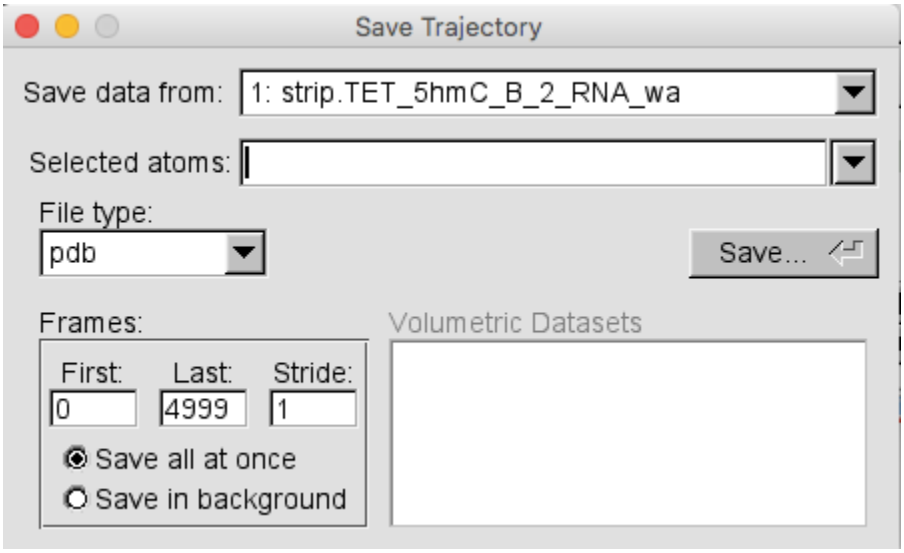
To open visualization states, either follow either follow `File → Load Visualization State` in the main menu, or type:

```
$ vmd -e name-of-saved-state.vmd
```

when loading VMD from the command line.

Saving Files

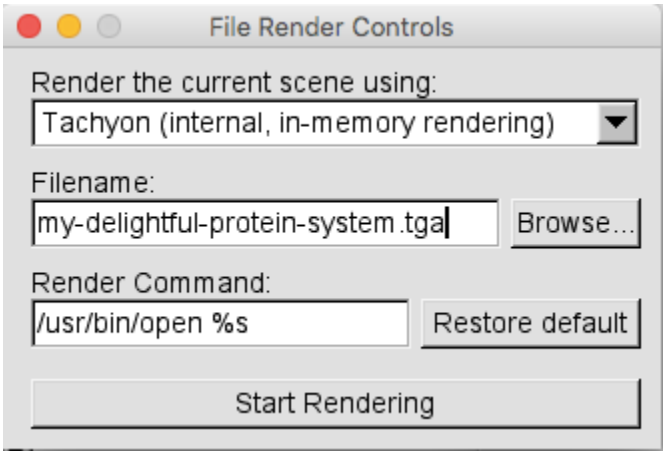
VMD is pretty useful for saving structures. For instance, you can save the final frame of a trajectory (so a loaded in `prmtop` and `nc` file combo), and use that final structure for all of your images. Pretty cool, huh? No? Alright, tough crowd. Anyway, you can do that by first highlighting the structure you'll want to save something from in the main menu, then following `File → Save Coordinates`. When saving a PDB from the prmtop/nc combo, then you're going to want to change the first frame to be equal to the last frame value (so in the image below, both should be 4999), because otherwise the PDB will contain a ridiculous number of coordinates and thus won't actually be helpful. You can also use VMD's syntax (page 63) to specify which things you want saved (usually `all not water`). Also, with frame numbers, as you can see, VMD starts a "0" and counts upwards, so while there were 5000 frames loaded, it'll register as 0 to 4999.



The Save Trajectory menu.

Generating Images

If you've been reading in order, then you know that I keep throwing this "publication-quality images" phrase around. Well, what do you know, the "Generating Images" section will actually tell you how to make these images! Amazing. Once you've gotten the the screen to look how you want the image you wanted created to look, follow `File → Render` in the main menu. This brings up the `File Render Controls` menu (see the image below).



The Render screen for image generation.

The first dropdown, `Render the current scene using:`, selects the image-quality type. If you're just doing something super quick that's kind of irrelevant, then it's fine to save the file with the default `Snapshot (VMD OpenGL window)`. If you're going to want to use the image for a presentation, poster, or publication, however, then you'll want to select `Tachyon (internal, in-memory rendering)`. For a juxtaposition of these, see the next to figures. (We'll get to why the colors are different at the end of this section.)

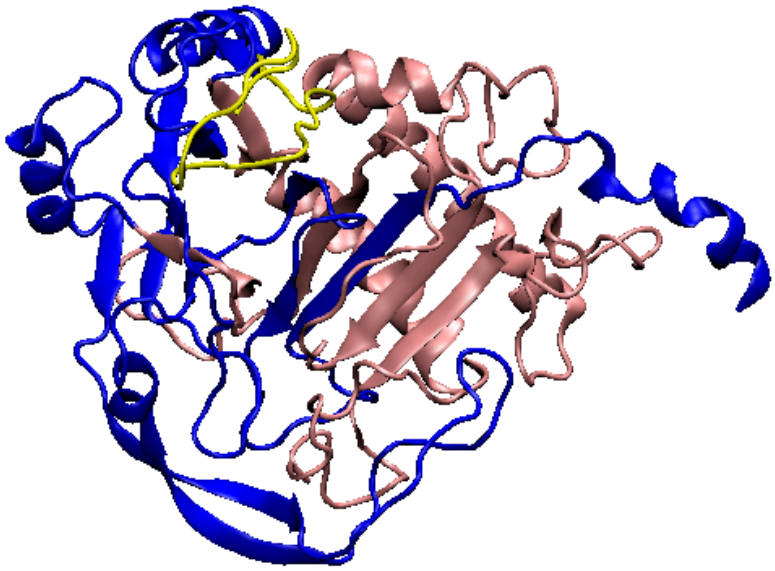


Image generated with Snapshot (VMD OpenGL window).

Image generated with Tachyon (internal, in-memory rendering).

The second dropdown is where you select the save location and the filename. For these images, keep the `.tga` extension; you can convert the `.tga` using Gimp or the UNIX `convert` (page 0) command later. The final box shouldn't be something you have to mess with at all, as it is the default option. Once you're ready, hit `Start Rendering`.

Woohoo! You've saved a picture! Enjoy remaking it 73 more times because it wasn't good yet. Oh, and a quick note: If you're coloring specific sections of a protein (like with `resid 1 to 23` and `resid 95 to 293`), then you will need to make sure that you don't also have just `protein` selected; it'd need to be like `protein not resid 1 to 23 and not resid 95 to 293`. The reason why is because VMD cannot handle the layering in image creation, and your picture will look terrible (just revisit the images above).

Saving a High-Quality Image with Transparency

You can also save higher-quality images in VMD (with transparency!) using POVRay . It needs to be installed separately.

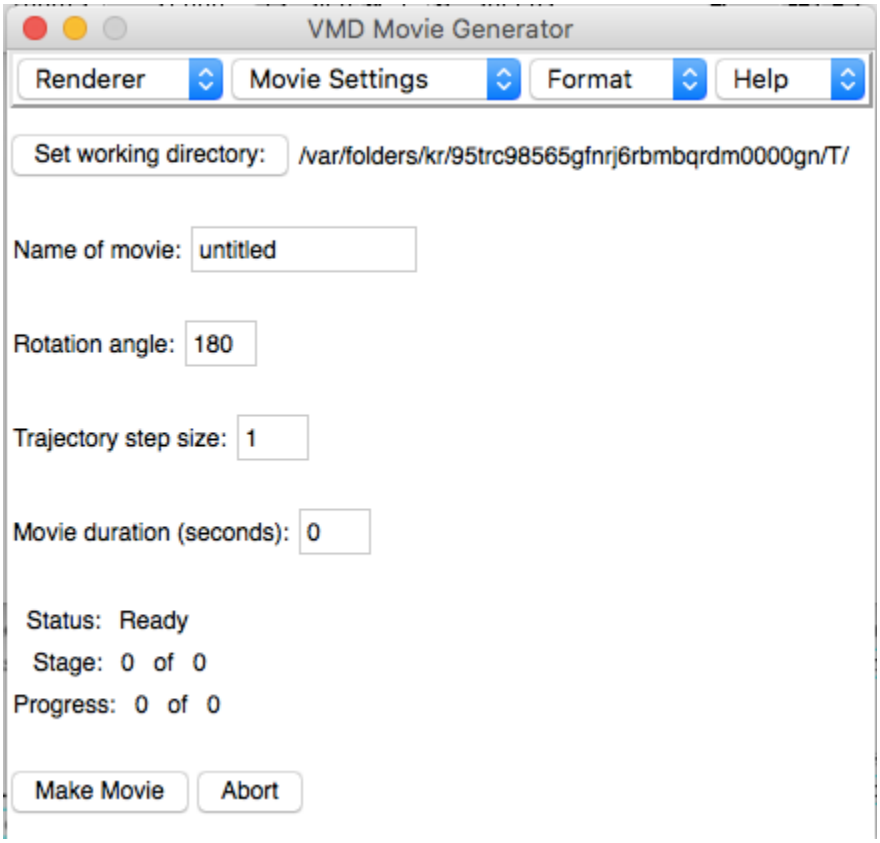
The ray-tracers work well with the AO-type and diffuse materials. Select `POV-Ray` in the render window. The number version (likely 3.6) is the earliest version you can use. For the render command, you need the `+UA` option to make it transparent. The image renders using the width and height of the on-screen window. Because you want a higher quality image, you can add a `0` after the height and width. It will take a bit more time (almost a full minute for a 450 residue protein), but it looks much better than 300 DPI.

When you specify the file command, you may wish to leave off the `.pov` extension. That way, when it renders, the final image file isn't named like `.pov.tga`.

- For TGA output: `povray +W%w0 +H%h0 -I%s -O%s.tga +X +A +FT +UA`
- For PNG output: `povray +W%w0 +H%h0 -I%s -O%s.png +X +A +FN +UA`

Making Movies

Now that we've talked about images, it's also important to discuss making movies. Yep, you read that right–you can save your protein's wiggles in a video! To do this, follow `Extensions → Visualization → Movie Maker` in the main menu. This brings up the `VMD Movie Generator` menu (see the image below).



The VMD Movie Generator menu.

The first thing you should do is choose the working directory for making the movie. It's set as a temp folder by default, which isn't necessarily a bad thing, but it does mean that your movie will be saved in that temporary directory. So set it to a place you can find it, or just know how to get to that default temp directory. Next, title your film! Make it descriptive so future you knows what system it is and what you're looking at.

Up in the dropdown menus, the first up is the `Renderer`. Unlike images, using the `Snapshot (screen capture)` option is acceptable for movies. However, if you chose this option, it is critical that the VMD window isn't covered by other windows (or browsers or terminals, etc.) in the process, because otherwise your video will have those landmarks in it. Under `Movie Settings`, there are two that

you'll likely want to use for movies. The first is `Rotation about Y axis` –this does exactly like it sounds. The second is `Trajectory`. This is what will actually save your wiggling protein as a movie. As for format, the default `MPEG-1 (ppmtompeg)` should be fine, unless you have a strong reason for needing a different format.

The other options, `Rotation angle`, `Trajectory step size`, and `Movie duration` depend on what you're doing and what you're making the movie for. The rotation angle specifies how far anything that rotates should. If you're using `Rotation about Y axis` to show off your protein, then you probably want that to be `360`; if you're making a trajectory movie then you'll want it to be `0`. `Trajectory step size` specifies how many frames to skip when making your movie. The larger the step size, the more choppy the video will look, but the shorter it will be. You only can choose to set a trajectory movie based on step size or duration–not both. Typically, people won't want to watch anything more than a 25 second video played on a loop.

Once you've decided all the settings, hit that `Make Movie` button, making sure that nothing is in the way of the VMD OpenGL Display window (where you are currently looking at the protein), and watch it go.

Success! A video has been made!

NMA Overview

Normal modes describe the vibrations of different molecular compounds. Analyzing these different modes can be done through the use of VMD and ProDy.

You can think of normal modes through the act of breathing. There are a lot of different movements involved in the act of breathing. For example, your lungs expand, your diaphragm moves down, and your mouth opens wide. Each of these motions have a different contribution to the act of you breathing. The most dominant, your lungs expanding, would be the primary mode (mode 1)–it's the biggest contributor to the act of breathing. The other motions are different modes (e.g., mode 2 or mode 3). You can break down breathing into hundreds or thousands of motions, but after those first few major ones, there's not much else that's important (i.e., any movement to your left pinky toe when you breathe probably isn't relevant).

Downloading VMD and ProDy

Visual Molecular Dynamics (VMD) has already been installed on the lab computers, but you can also install it on a personal computer from here for free.

ProDy is a plugin compatible with VMD. To install ProDy on a computer with pip installed, use `pip install –U ProDy`. Otherwise, instructions are available from here .

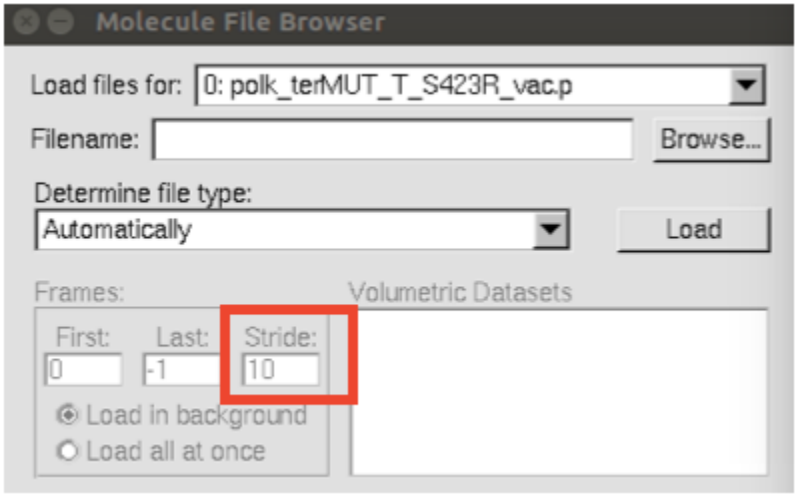
To use ProDy, two Python dependencies are required (*biopython* and *numpy*). If you have an Anaconda python installation, these can be installed with `conda install biopython` and `conda install numpy`. Otherwise, installing they can be installed the same way with pip.

Loading in the Protein

For the normal modes analysis, you need to have previously finished the *cpptraj* steps to create trajectory file for all the production steps. The topology file can be opened with vmd from the command line, but because the coordinate file is really large, it needs to be loaded in from the user interface. When loading in the data for the prmtop, set stride equal to 10 so that it only loads every 10 frames (see the image below (page 75)).

```
$ vmd -prmtop vacuum.prmtop
```

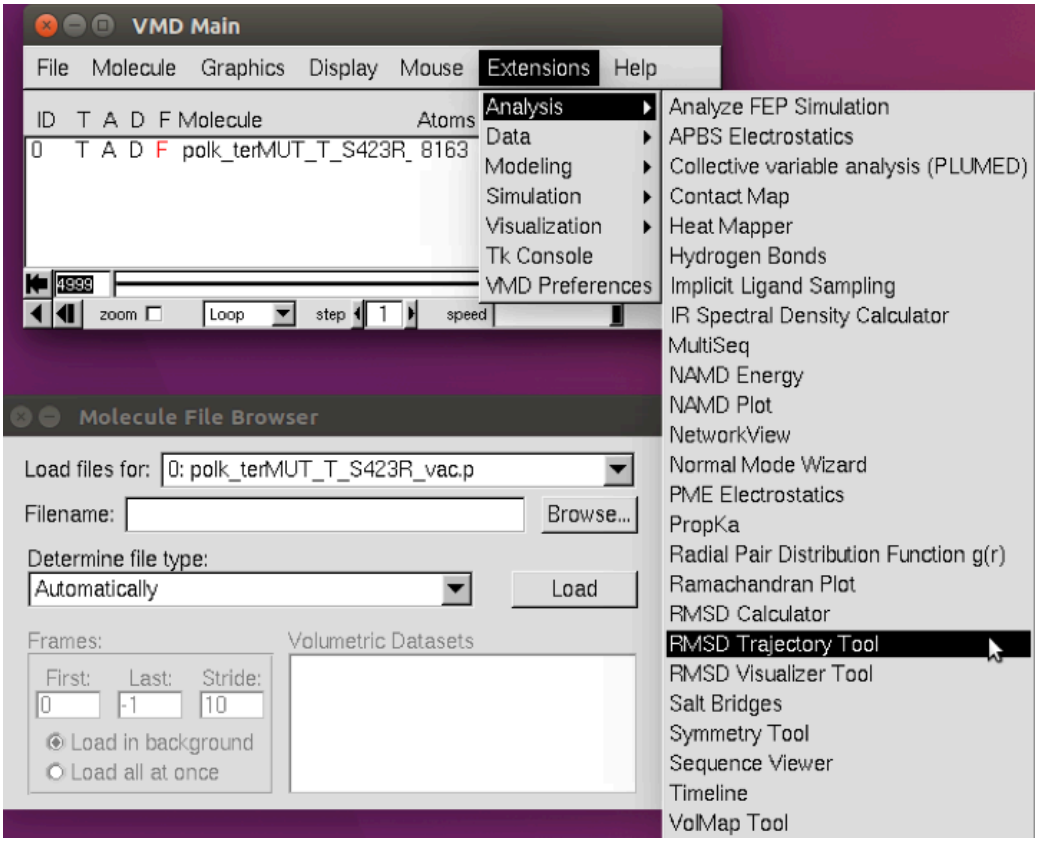
Setting Stride



Demonstration of where to set Stride to skip frames when loading the coordinate file.

Once the trajectory files are loaded in, follow the VMD menu through `Extensions → Analysis → RMSD Trajectory Tool` (see below (page 76)).

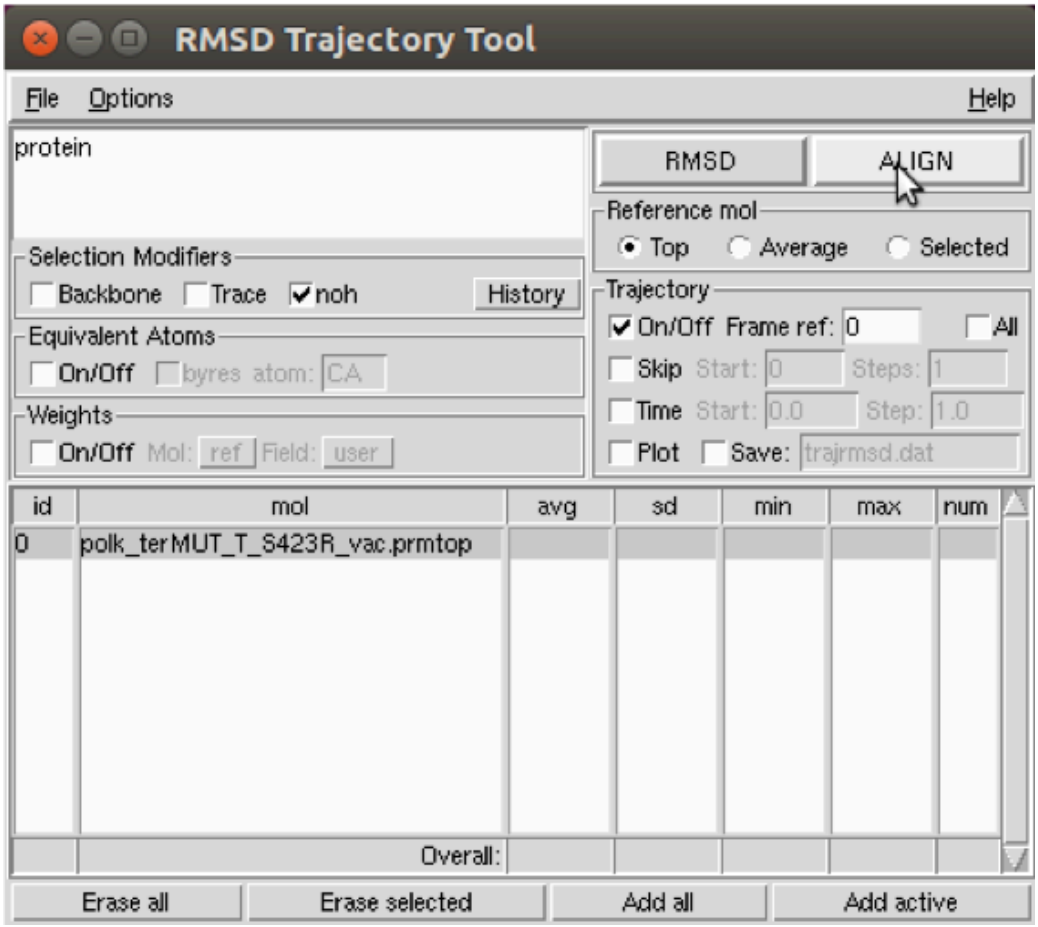
RMSD Analysis



Location of RMSD Trajectory Tool.

This brings up the menu (shown below (page 77)) with several options. Click align.

RMSD Trajectory Tool

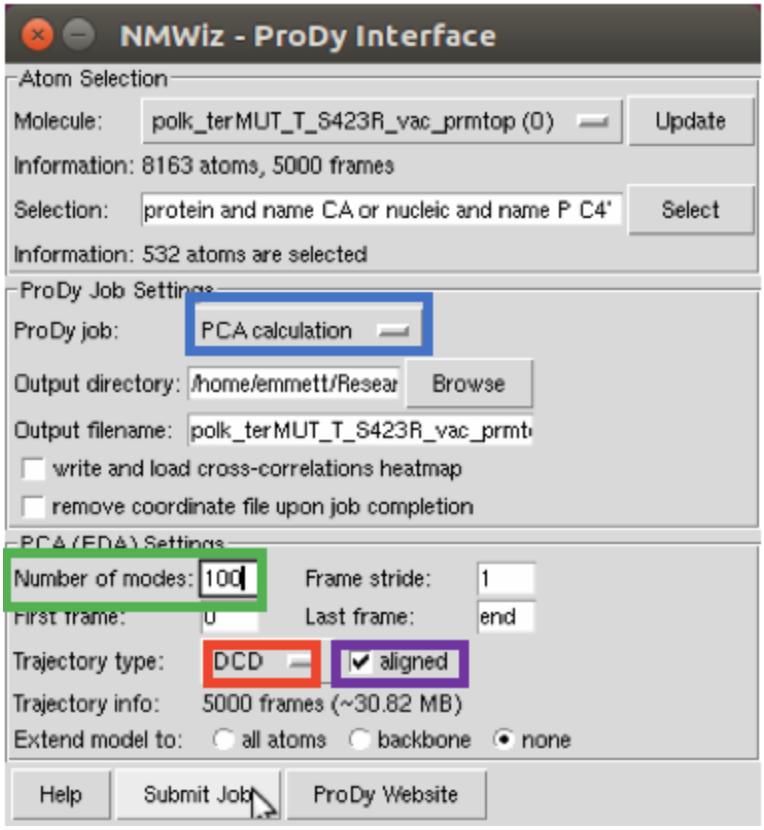


The RMSD Trajectory Tool window, with the cursor over align.

After the structure is aligned, proceed to the Normal Mode Wizard. It can be found in the same menu as the RMSD Trajectory Tool, shown previously (page 76).

From the NMWiz menu, select ProDy Interface. This pulls up the ProDy window (page 78). Several things need to be changed in this interface, including setting the ProDy job to a PCA (Principle Component Analysis) calculation, changing the number of modes to 100 (it can analyze every mode, but the only important ones are the first few), changing the trajectory type to DCD, and clicking the aligned structure option. The aligned structure option is what was performed using the RMSD Trajectory Tool, and saves time in the normal mode calculation. After changing each of these, the job is ready to be submitted.

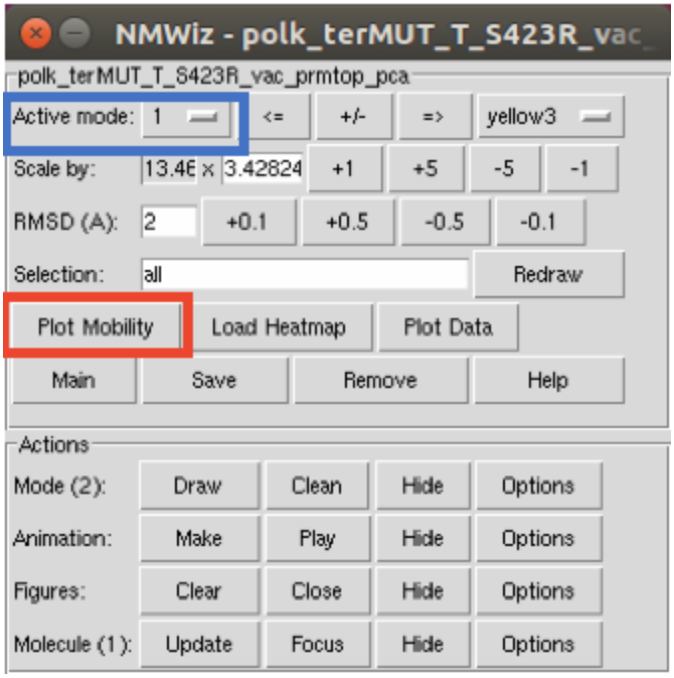
ProDy Interface



The ProDy Interface window. The necessary changes are emphasized with their corresponding colors: blue-ProDy job, green-number of modes, red-trajectory type, purple-aligned.

Once submitted, a new NMWiz window appears (see below (page 79)). The mode that is being visualized appears in the upper left. From this window (and several successive windows), you can save the information for the normal nodes for further analysis and comparison. To do this, select Plot Mobility.

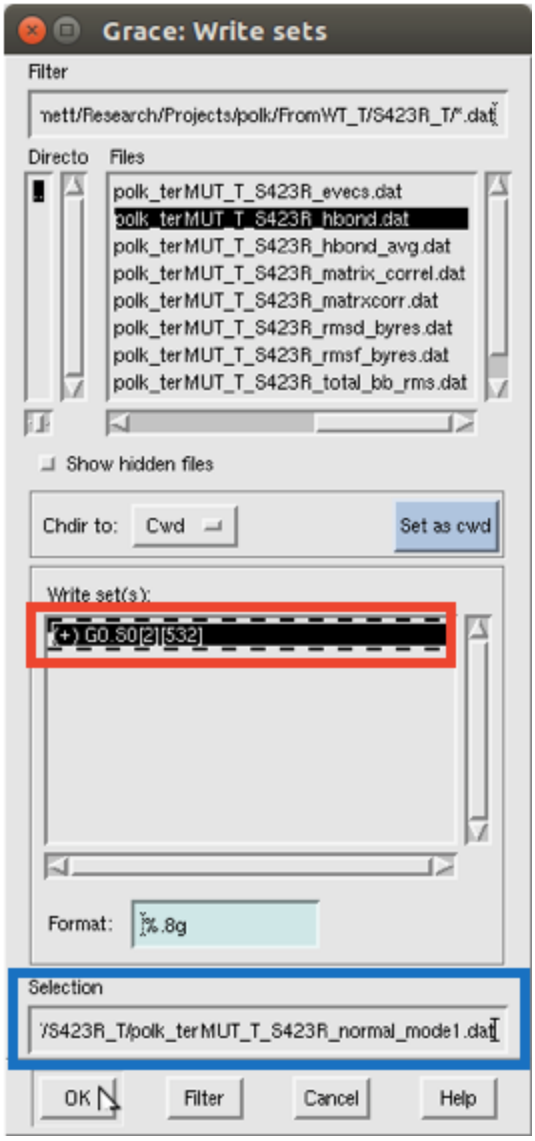
NMWiz Window



The next NMWiz window, with the blue box corresponding to how to change between modes, and the red box corresponding to the Plot Mobility button.

Plot Mobility brings up a plot (amazing!). The data from this plot need to be saved in a roundabout way. To start this process, go to `File → Save as xmgrace`. Title this file however you want, but recognize this will be done for more than 1 mode. Once saved, the Grace window will appear. In this window, follow `Data → Export → ASCII`. Yet another Grace window will appear, allowing you to save the data as a `.dat` file (see below (page 80)). To actually save, first title the new file (under the selection box), and then click on the set in “Write Sets” that you want to save. Then, click OK. It won’t look like it did anything, so if you’re unsure if you clicked it, click OK again and it will ask if you want to overwrite the file.

xmgrace Window



The Grace window that allows you to save the normal mode data as a .dat file. In red is the set of data that should be selected and in blue is the save location.

Follow this process for the first 3 normal modes (1, 2, and 3).

Plotting Normal Modes with gnuplot

Gnuplot is a freely available plotting utility.

The following (`normalmodeplot.gnu`) is an example gnuplot script that can be used to generate graphs of the first three normal modes for a specific system. This script would be in a directory that contained the directories of the individual systems (4 plots are being generated with this one script as-written), which are then accessed individually in the plot command. The individual .dat files are the ones that were created using NMWiz and ProDy (page 75). The number of residues should be changed in the `set xrange [0:500]` line to reflect the number of residues of the protein, and the `set x2tics` line should have **"DNA" 400** reflect the residue number where DNA actually begins (if there is any; otherwise delete it or comment it out).

```
set encoding iso_8859_1
set term postscript eps enhanced color font "Helvetica, 20";

unset ytics
set xlabel "Residue Number"
set ylabel "PCA Square Fluctuations"
set xrange [1:500]
set xtics border nomirror out
set x2tics border nomirror out rotate by 25 ("DNA" 400)
set boxwidth 0.25
set style fill solid

set output "WT_protein_system_modes.eps";
plot "WT-system/WT-system_mode1.dat" u 1:2 w boxes t "Mode 1" l
w 3, \
"WT-system/WT-system_mode2.dat" u 1:2 w boxes t "Mode 2" lw 3,
\
"WT-system/WT-system_mode3.dat" u 1:2 w boxes t "Mode 3" lw 3;

set output "MUT-A-system_modes.eps";
plot "MUT-A-system/MUT-A-system_mode1.dat" u 1:2 w boxes t "Mod
e 1" lw 3, \
"MUT-A-system/MUT-A-system_mode2.dat" u 1:2 w boxes t "Mode 2"
lw 3, \
"MUT-A-system/MUT-A-system_mode3.dat" u 1:2 w boxes t "Mode 3"
lw 3;

set output "MUT-B-system_modes.eps";
plot "MUT-B-system/MUT-B-system_mode1.dat" u 1:2 w boxes t "Mod
e 1" lw 3, \
"MUT-B-system/MUT-B-system_mode2.dat" u 1:2 w boxes t "Mode 2"
lw 3, \
"MUT-B-system/MUT-B-system_mode3.dat" u 1:2 w boxes t "Mode 3"
lw 3;

set output "MUT-C-system_modes.eps";
plot "MUT-C-system/MUT-C-system_mode1.dat" u 1:2 w boxes t "Mod
e 1" lw 3, \
"MUT-C-system/MUT-C-system_mode2.dat" u 1:2 w boxes t "Mode 2"
lw 3, \
"MUT-C-system/MUT-C-system_mode3.dat" u 1:2 w boxes t "Mode 3"
lw 3;
```

Determining Normal Modes with Python

Typically, the first three normal modes are the most important for a system, as they contribute the greatest to the overall motion of the protein. However, some cases occur where they aren't that informative, i.e. there's so much different movement occurring that those three are not distinct enough. You can plot the different contributions using Python to get a clearer picture regarding different contributions.

The following is `eigenplots.py` and can be run with `python eigenplots.py`. Thanks, Alice!

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import statsmodels.api as sm
from matplotlib.mlab import griddata
from tables import *
from matplotlib.colors import LinearSegmentedColormap

data1 = np.genfromtxt("WT-system/WT_protein_system_vacuum_prmtop_pca.nmd", delimiter=None,skip_header=9)

plt.rcParams.update({'font.size': 22})
plt.rcParams.update({'figure.autolayout': True})

data1eigenrank = data1[:,1]
data1eigenvalue = data1[:,2]
plt.tight_layout()
#x-axis 0 to 10; y-axis 0 to 50
plt.axis([0,10,0,50])
plt.xlabel('Mode Number')
plt.ylabel('Percentage of\nTotal Motion (%)')
plt.plot(data1eigenrank,data1eigenvalue,marker='o',c='black',linewidth=2.0)
plt.savefig('WT_protein_system_eigenplot.png')
plt.gcf().clear()
```

First a bunch of things were imported to let Python know it knows how to use Python. Then, the dataset is found, and several unhelpful lines are skipped. The auto-plot parameters and font sizes are reset, so the graph isn't weirdly smushed. Then two specific things are pulled from the matrix in order to be plotted, and the plot information is specified.

Fast NMA (ft. cpptraj and Python)

The NMD file can be prepared for use with ProDy using *cpptraj*. How cool is that?

First, you need to create a covariance matrix using the `matrix` command. Covariance is a fancy way to say that you're looking for the overall change in A with respect to B, B with respect to C, and so on. In this case, we're trying to reduce down multi-dimensional coordinate data and then compare each of the residues to each other.

After creating the matrix, use the `diagmatrix` command to calculate the eigenvectors of the matrix and generate the NMWiz files.

```
## For normal modes (evecs = eigenvectors)
matrix out WT_protein_system_covar_mat.dat name norm_mode :1-47
6@CA,P,C4',C2 covar
diagmatrix norm_mode out WT_protein_system_evecs.out vecs 100 r
educe \
 nmwiz nmwizvecs 100 nmwizfile WT_protein_system_100.nmd nmwizm
ask :1-476@CA,P,C4',C2
```

The mask used above (`@CA,P,C4',C2`) is based on what the ProDy interface in VMD uses (`protein and name CA or nucleic and name P C4' C2`) by default.

Python can be used to make the mobility plots that became the normal mode plots (page 83) that we got to know and love with gnuplot. Future Mark will share a script for that on his GitHub . Until then, I have included my forked version here.

NMA_plot_mult.py

```
import numpy as np
import prody as prd
import matplotlib.pyplot as plt

## Typically 3 modes will be enough
num_of_modes = 4

## Create a list of tuple (infile_name, outfile_name, system_na
me)
## These are the NMD file, the PNG file, and the tag for the sy
stem to determine
## the top X ticks through plot_ticks
in_out_sys_names = [
  ("WT_protein_system_r1_100.nmd", "WT_protein_system_r1_NMA.pn
g", "WT"),
  ("WT_protein_system_r2_100.nmd", "WT_protein_system_r2_NMA.pn
g", "WT"),
  ("WT_protein_system_r3_100.nmd", "WT_protein_system_r3_NMA.pn
g", "WT"),
  ("MUT_A_system_r1_100.nmd", "MUT_A_system_r1_NMA.png", "MUT
A"),
  ("MUT_A_system_r2_100.nmd", "MUT_A_system_r2_NMA.png", "MUT
A"),
  ("MUT_A_system_r3_100.nmd", "MUT_A_system_r3_NMA.png", "MUT
A"),
  ("MUT_B_system_r1_100.nmd", "MUT_B_system_r1_NMA.png", "MUT
B"),
  ("MUT_B_system_r2_100.nmd", "MUT_B_system_r2_NMA.png", "MUT
B"),
  ("MUT_B_system_r3_100.nmd", "MUT_B_system_r3_NMA.png", "MUT
B"),
]

def plot_ticks(sys, NMA_data):
    """Sets top xticks. You NEED the 0 and NMA_data.numAtom
s(), otherwise the
    scale will be turned off.
    **This is an example, you'll need to modify it for your sys
tem.**
    """
    if sys == "MUTA":
        labels_top = ["", "MUTA", "GS linker", "", "DNA", ""]
        places_top = [0, 141, 334, 346, 431, NMA_data.numAtom
s()]
    elif sys == "MUTB":
```

```
        labels_top = ["", "GS linker", "", "MUTB", "DNA", ""]
        places_top = [0, 334, 346, 378, 431, NMA_data.numAtom
s()]
    elif sys == "WT":
        labels_top = ["","GS linker", "", "DNA", ""]
        places_top = [0, 334, 346, 431, NMA_data.numAtoms()]
    else:
        labels_top = []
        places_top = []
    return labels_top, places_top


def NMA_plots(filename,outfile,sys):
        """Creates a plot of the most important modes for a sys
tem.
        Parameters
        ----------
        filename : str
            An NMD file.
        outfile: str
            Name of the output PNG.
        sys: str
            A name (e.g., WT, MUT A, MUT B, etc.) for the syste
m. Replicates
            should have the same name.
        """
        NMA_data,Atom_Group = prd.parseNMD(filename)
        eigens = NMA_data.getEigvals()

        labels_top, places_top = plot_ticks(sys, NMA_data)

        scales=[]
        temp = open(filename)
        lines = temp.readlines()
        temp.close()
        for line in lines:
            if 'mode' in line[:5]:
                scales.append(float(line.split()[:3][-1]))

        ## Make an array of the number of atoms for plotting
        x_vals = np.arange(0, NMA_data.numAtoms(), 1)

        fig = plt.figure(figsize=(10,8),dpi=300)
        ax = fig.add_subplot(1,1,1)
        for i in range(num_of_modes):
```

```
            dataset = [np.linalg.norm(NMA_data.getEigvec
s()[:,i][n:n+3])*scales[i]*eigens[i] for n in range(0, NMA_dat
a.numEntries(), 3)]
            ax.bar(x_vals, dataset, width=1.0, label="Mode "+st
r(i+1))

        ax_top = ax.twiny()
        ax_top.set_xticks(places_top)
        ax_top.set_xticklabels(labels_top, fontdict=None, mino
r=False)

        ax.legend()
        ax.set_xlabel("Residue Number")
        ax.set_ylabel("PCA Square Fluctuations")
        ax.set_xlim([0,x_vals.size]) ## Remove white space at e
dge
        plt.tight_layout()
        fig.savefig(outfile,dpi=300)
        plt.close()

for filename,outfile,sys in in_out_sys_names:
    NMA_plots(filename,outfile,sys)
```

Reading the `.nmd` File with Structure in VMD

Reading the `.nmd` file can be done on its own with the `NMWiz` plugin, but that file doesn't contain structure information that you probably want for image making.

This can be addressed through the following steps:

1. Load in either a PDB file or both a `.prmtop` and `.rst` file into VMD.
2. Follow `Extensions → Analysis → Normal Mode Wizard → Load NMD File` to load the NMD file.
3. Open the RMSD Trajectory Tool through `Extensions → Analysis → RMSD Trajectory Tool`.
4. In the Tool, match the name you used in `cpptraj` to create the command in the top left box (likely `name CA P "C4'" C2`).
5. In the `Trajectory` box on the right, deselect all options.
6. Remove the `{ default_name arrow}` option by highlighting and choosing `Erase selected`.
7. Choose the `Selected` bubble under `Reference mol`.

8. Highlight `{ default_name coordinates}` and click `RMSD`.
9. Once the RMSD has finished, click `ALIGN`.

You specifically want to highlight `{ default_name coordinates}` so that the arrows and “coordinates” stay in the same place and only the full structural coordinates move. Otherwise, the tube tracing and stucture will be in one place and the arrows will be in another. This transformation will not be saved in a `.vmd` visualization state file.

Using the Trajectory Tool

The RMSD Trajectory Tool for VMD. The top left has the box for selections, the top right has the RMSD and ALIGN buttons. Below them is the Reference mol section, and below that trajectory options.

VMD's RMSD Trajectory Tool.

EDA Overview

Energy decomposition analysis (EDA) is a quantitative means of understanding chemical bonds. The way that we go about EDA provides information about the Coulomb and van der Waals energies for different residues of interest.

We go about performing EDA using a Fortran90 program adapted by Dr. Cisneros . The program uses an input file (described here (page 92)) that includes some specific information about the system being studied, including the number of residues in the associated mdcrd file. One thing to note for the Fortran90 program is that the input files must be in ASCII format. Starting with AMBER16, the default format for all generated files is NetCDF and not ASCII; unless you explicitly set `ioutfm=0` in your mdin files, you will have them in NetCDF format. To check if your mdcrd is in ASCII format, use something like `head _____.mdcrd`. If you see neat rows of numbers, then it is in ASCII. If it looks like your computer is exploding in the Terminal, then you'll need to use cpptraj to create an ASCII formatted mdcrd. Another (easier) way to do this is to use `file _____.mdcrd`. If the file is a NetCDF, then `_____.mdcrd: data` would be returned; otherwise `_____.mdcrd: ASCII text` will be returned. You can use *cpptraj* to convert to ASCII through adding

```
trajout _____.mdcrd crd
```

to the end of your a *cpptraj* input (that includes all the trajin lines for files of interest). It is critically important that any time cpptraj is used to convert to ASCII format that no strip commands are used. Stripping the system, while saving on time, will return radically different values for both Coulomb and van der Waals energies. Using autoimage will have an effect on the significant figures, so it should also not be used.

Once the Fortran90 program has run (correctly), there are 3 output files generated. These are fort.804, fort.803, and fort.806. The first one, `fort.804`, is a sanity check that may include the atom typing. It is created at the start of the program execution. The other two files, `fort.803` and `fort.806`, are created after the program has finished. `fort.803` includes information about the Coulomb energies; `fort.806` includes information about the van der Waals energies. For both of these files, the columns are organized from left to right as:

1. Index (row number)
2. Residue A
3. Residue B

4. Coulomb or van der Waals energy (in kcal mol^{-1})
5. Standard error

EDA Input File

EDA uses an input file that consists of the number of protein residues, the number of files to be read in, the total number of atoms, protein atoms, total residues, and number of types. The final line should be the names of the mdcrds (in ASCII format) that the program should use. The file extension on the input file should be `.inp`.

All of the associated numbers can be found from looking at a PDB file for the system. The final line of the protein residues and the final line of the solvated system from which the first example input was generated is shown here, with the relevant numbers in brackets.

```
ATOM   [7277]  O4  AKG    [455]     3.870  -2.076   2.583  1.0
0  0.00
ATOM  [59661]  H2  WAT  [17929]    -1.189   1.046 -55.340  1.0
0  0.00
```

This example is for a protein that the strip command was not used on.

```
455 !number of protein residues
1 !number of files
59661 !total number of atoms
7277 !number of protein atoms
17929 !number of total residues
2000 !max number of types
solvated_complex_imaged_1-200-full.mdcrd
```

This example is for a protein that the strip command was used on. Notice that the protein and the residue totals are equivalent. Using strip will return bad values, so it shouldn't be used.

```
455 !number of protein residues
1 !number of files
7278 !total number of atoms
7278 !number of protein atoms
455 !number of total residues
2000 !max number of types
solvated_complex_imaged_1-200.mdcrd
```

This example shows what it looks like when multiple mdcrd files are read in.

```
473 !number of protein residues
10 !number of files
59663 !total number of atoms
7295 !number of protein atoms
17929 !number of total residues
2000 !max number of types
solvated_complex_md1.mdcrd
solvated_complex_md2.mdcrd
solvated_complex_md3.mdcrd
solvated_complex_md4.mdcrd
solvated_complex_md5.mdcrd
solvated_complex_md6.mdcrd
solvated_complex_md7.mdcrd
solvated_complex_md8.mdcrd
solvated_complex_md9.mdcrd
solvated_complex_md10.mdcrd
```

Locally Running EDA

The EDA program can be compiled and run locally at your workstation. To achieve this with a program edition titled `Residue_E_Decomp_07_15.f90`, perform:

```
$ gfortran Residue_E_Decomp_07_15.f90 -o Residue_E_Decomp_07_1
5.x
$ ./Residue_E_Decomp_07_15.x
```

The program will look for an input file (ex: here (page 92)) and whatever prmtop file should be used with your specified mdcrd. If you use a strip command in your *cpptraj* ASCII generation, then you'll need a stripped prmtop (either made with the `outprefix` option of the strip command or generated through the *parmed* (page 24) program in AmberTools).

EDA PBS Script

The following script can be used to run EDA on a PBS scheduler (page 0). The script can be amended to work with a different Fortran compiler (here, Intel's ifort is used).

```
#/bin/bash
#PBS -q gac.cpu
#PBS -j oe
#PBS -r n
#PBS -o EDA.error
#PBS -N EDA-run-script

##Load in the Intel compiler
module load intel/17.0

##Access the folder where the files are
cd $PBS_O_WORKDIR

##Compile the EDA program
ifort Residue_E_Decomp_07_15.f90 -o Residue_E_Decomp_07_15.x

##Sleep for 5 seconds, ensuring that the program was compiled
sleep 5

##Run the program; read in the prompt answers  [Line 1: Name o
f input; Line 2: Name of prmtop]
./Residue_E_Decomp_07_15.x < ans.txt

##Acquire the process ID for the program execution
proc_PID=$!

##Wait until the program execution is over before ending the sc
ript
wait $proc_PID

echo "All done!"
```

A file named `ans.txt` is fed into the Fortran program. Because the program is being run through the queue, the answers to the prompts must be directly input into the program. And example of the file is:

```
EDA-input.inp
solvated_complex.prmtop
```

If any comments are used in the `ans.txt` file, the program will not work.

Interactive EDA Submission

Instead of using a script, the EDA program can be run through the PBS scheduler interactively. First, request a node for interactive submission with:

```
$ qsub -I -q my_cpu_alloc -N name-of-job-for-queue
```

After requesting a node, you can essentially follow through the start of the EDA script.

```
$ module load intel/17.0
$ cd /home/euid123/path/to/files
$ ifort Residue_E_Decomp_07_15.f90 -o Residue_E_Decomp_07_15.x
$ ./Residue_E_Decomp_07_15.x
        Answer prompt 1 [Name of input file?]
        Answer prompt 2 [Name of prmtop?]
        Cntrl + Z
$ bg
$ top
```

The last three steps (`Cntrl + Z` through `top`) ensures that any ssh session used to run the program won't terminate due to a broken pipe. The job is temporarily suspended, forced to run in the background, and `top`, which refreshes every 5 seconds, is used to keep the connection alive. When the program is no longer listed on `top`, then you can exit the node/ssh connection.

Like before (page 95), Intel's ifort compiler doesn't need to be used; gcc's gfortran can be used instead.

EDA Results Analysis with R for Specific Residues

As mentioned earlier (page 10), R is a programming language that is often used for data processing and statistics. The following script has been created to process through replicate EDA data and obtain the averages for a system.

rmagic-EDA-avg.r

```
## Run this with "Rscript rmagic-EDA-avg.r"
## (Assuming you've already installed R...)

#-----------------------------------------#
#--Specify the paths to the Files from EDA--#
#-----------------------------------------#

## This script has been pre-built for a system with 3 replicate
s
## More or less than 3 reps (up to 5) can be achieved through
## Commenting or uncommenting

## Paths to the fort.803 (Coul) files
## Set A (system A)
infile1Ac <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/WT-System-1/fort.803")
infile2Ac <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/WT-System-2/fort.803")
infile3Ac <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/WT-System-3/fort.803")
##infile4Ac <- Sys.glob("/absolute/path/to/the/analysis/files/f
or/WT-System-4/fort.803")
##infile5Ac <- Sys.glob("/absolute/path/to/the/analysis/files/f
or/WT-System-5/fort.803")

## Paths to the fort.806 (VdW) files
## Set A (system A)
infile1Av <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/WT-System-1/fort.806")
infile2Av <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/WT-System-2/fort.806")
infile3Av <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/WT-System-3/fort.806")
##infile4Av <- Sys.glob("/absolute/path/to/the/analysis/files/f
or/WT-System-4/fort.806")
##infile5Av <- Sys.glob("/absolute/path/to/the/analysis/files/f
or/WT-System-5/fort.806")

#----------------------------#
#--Define your outfile names--#
#----------------------------#

## A is for infiles labeled A
## Each system gets an averaged file
## Have one for Coulomb, one for vdW, and one for Coul+vdW (tot
```

```
al)

A_coul <- "/absolute/path/to/the/avgeraging/output/WT_protein_s
ystem_EDA_resX_coul_avg.dat"
A_vdw <- "/absolute/path/to/the/avgeraging/output/WT_protein_sy
stem_EDA_resX_vdw_avg.dat"
A_tot <- "/absolute/path/to/the/avgeraging/output/WT_protein_sy
stem_EDA_resX_tot_avg.dat"

## Residue of interest (A matched with B, which matches do you
care about?) (use 4 after decimal)
## Ex. If mutant is residue 100, this would be 100
## This script will remove the matches directly surrounding RO
I for you
## Which is good because by not being just Coul and vdW, they'r
e too dominant
ROI <- 100

## How many data sets to add (use 4 after decimal)
sets <- 3.0000
#sets <- 5.0000

#-----------------------------------------------------------------
--------#
#---------Behind the Curtain: No Need to Modify Past This Lin
e---------#
#-----------------------------------------------------------------
--------#

## Use the data tables package to read in data frames
## Remove comment to install locally
#install.packages("data.table")
library(data.table)

## Use the tidyverse package to perform string replacement
## Remove comment to install locally
#install.packages("tidyverse")
library(tidyverse)

## Turn off scientific notation
options(scipen = 999)

#---------------------------#
#--Read in Coul EDA Scripts--#
#---------------------------#
```

```
## First line of file is number of frames used for EDA
## This is skipped by R's fread by default to avoid
## Irregular header information

## Reading each file as a data.table.
## Bonus - fread is much faster than read.csv
read1Ac <- fread(infile1Ac, header=FALSE)
read2Ac <- fread(infile2Ac, header=FALSE)
read3Ac <- fread(infile3Ac, header=FALSE)
#read4Ac <- fread(infile4Ac, header=FALSE)
#read5Ac <- fread(infile5Ac, header=FALSE)

colnames(read1Ac) <- c("Index", "ResidueA", "ResidueB", "Coulom
b", "StdErr")
colnames(read2Ac) <- c("Index", "ResidueA", "ResidueB", "Coulom
b", "StdErr")
colnames(read3Ac) <- c("Index", "ResidueA", "ResidueB", "Coulom
b", "StdErr")
#colnames(read4Ac) <- c("Index", "ResidueA", "ResidueB", "Coulo
mb", "StdErr")
#colnames(read5Ac) <- c("Index", "ResidueA", "ResidueB", "Coulo
mb", "StdErr")

## Combine all the datasets into 1
bound <- rbind(read1Ac, read2Ac, read3Ac)
#bound <- rbind(read1Ac, read2Ac, read3Ac, read4Ac, read5Ac)

## Add in a blank row of the match for future plotting needs
extra <- data.frame(0, ROI, ROI, 0, 0)
bound <- rbind(bound, setNames(extra, names(read1Ac)))

#bound$Index <- as.numeric(bound$Index)
bound$Index <- as.numeric(bound$Index)
bound$ResidueA <- as.numeric(bound$ResidueA)
bound$ResidueB <- as.numeric(bound$ResidueB)
bound$Coulomb <- as.numeric(bound$Coulomb)
bound$StdErr <- as.numeric(bound$StdErr)

## Collapse repeat lines into themselves (i.e. add numbers toge
ther)
superbound_avg <- aggregate(data=bound, cbind(Coulomb,StdErr)~R
esidueA+ResidueB, FUN=sum)
superbound_sd <- aggregate(data=bound, cbind(Coulomb,StdErr)~Re
sidueA+ResidueB, FUN=sd)
```

```
## Get average based on number of sets combined [This if for 3]
superbound_avg$AvgCoulomb <- format(superbound_avg$Coulomb / se
ts, digits=4, format="f")
superbound_avg$AvgCoulombSD <- format(superbound_sd$Coulomb / s
ets, digits=4, format="f")

## If you for some reason care about StdErr, then you'd uncomme
nt this
## Yes, it's weird that StdErr has a SD, but that's just a sani
ty thing
#superbound_avg$AvgStdErr <- format(superbound_avg$StdErr / set
s, digits=4, format="f")
#superbound_avg$AvgStdErrSD <- format(superbound_sd$StdErr / se
ts, digits=4, format="f")

save_cols_Ac <- superbound_avg[,c("ResidueA", "ResidueB", "AvgC
oulomb", "AvgCoulombSD")]
#save_cols_Ac <- superbound_avg[,c("ResidueA", "ResidueB", "Avg
Coulomb", "AvgCoulombSD", "AvgStdErr", "AvgStdErrSD")]

only_ROI_rows_Ac <- filter(save_cols_Ac, ResidueA == ROI | Resi
dueB == ROI)

## Change the NA from standard deviation to 0
only_ROI_rows_Ac[(ROI),4] <- 0

## Create a copy of the parsed data to format
clean_rows_Ac <- data.frame(only_ROI_rows_Ac)

## Pattern searching converted it to a character string, so bac
k to numeric
clean_rows_Ac$AvgCoulomb <- as.numeric(clean_rows_Ac$AvgCoulom
b)
clean_rows_Ac$AvgCoulombSD <- as.numeric(clean_rows_Ac$AvgCoulo
mbSD)

## Limit to 4 sig figs after decimal
clean_rows_Ac$AvgCoulomb <- formatC(clean_rows_Ac$AvgCoulomb, d
igits=4, format="f")
clean_rows_Ac$AvgCoulombSD <- formatC(clean_rows_Ac$AvgCoulombS
D, digits=4, format="f")

## Set the two residues surrounding the ROI to zero
## This is because energy is overpowering due to other energy t
```

```
erms
## So if ROI=100, you remove matches between 99 & 100 as well a
s 100 & 101
if (ROI != 1) {clean_rows_Ac[(ROI-1),3] <- 0
clean_rows_Ac[(ROI-1),4] <- 0
}
clean_rows_Ac[(ROI+1),3] <- 0
clean_rows_Ac[(ROI+1),4] <- 0

#-----------------------------------------------------------------
---------#
#-------------------------COUL OUTFILE----------------------
#
#-----------------------------------------------------------------
---------#

## Now write a tab-delimited outfile!
## Don't care about the index rownames
#write.table(clean_rows_Ac, file = A_coul, sep="\t", row.name
s=FALSE, quote=FALSE)

## Write a whitespace-delimited outfile!
sink(A_coul, type=c("output"))
print(clean_rows_Ac, row.names=FALSE)
sink()

#-------------------------#
#--Read in VDW EDA Scripts--#
#-------------------------#

## First line of file is number of frames used for EDA
## This is skipped by R's fread by default to avoid
## Irregular header information

## Reading each file as a data.table.
## Bonus - fread is much faster than read.csv
read1Av <- fread(infile1Av, header=FALSE)
read2Av <- fread(infile2Av, header=FALSE)
read3Av <- fread(infile3Av, header=FALSE)
#read4Av <- fread(infile4Av, header=FALSE)
#read5Av <- fread(infile5Av, header=FALSE)

colnames(read1Av) <- c("Index", "ResidueA", "ResidueB", "VdW",
"StdErr")
colnames(read2Av) <- c("Index", "ResidueA", "ResidueB", "VdW",
```

```
"StdErr")
colnames(read3Av) <- c("Index", "ResidueA", "ResidueB", "VdW",
"StdErr")
#colnames(read4Av) <- c("Index", "ResidueA", "ResidueB", "Vd
W", "StdErr")
#colnames(read5Av) <- c("Index", "ResidueA", "ResidueB", "Vd
W", "StdErr")

## Combine all the datasets into 1
bound <- rbind(read1Av, read2Av, read3Av)
#bound <- rbind(read1Av, read2Av, read3Av, read4Av, read5Av)

## Add in a blank row of the match for future plotting needs
extra <- data.frame(0, ROI, ROI, 0, 0)
bound <- rbind(bound, setNames(extra, names(read1Av)))

#bound$Index <- as.numeric(bound$Index)
bound$Index <- as.numeric(bound$Index)
bound$ResidueA <- as.numeric(bound$ResidueA)
bound$ResidueB <- as.numeric(bound$ResidueB)
bound$VdW <- as.numeric(bound$VdW)
bound$StdErr <- as.numeric(bound$StdErr)

## Collapse repeat lines into themselves (i.e. add numbers toge
ther)
superbound_avg <- aggregate(data=bound, cbind(VdW,StdErr)~Resid
ueA+ResidueB, FUN=sum)
superbound_sd <- aggregate(data=bound, cbind(VdW,StdErr)~Residu
eA+ResidueB, FUN=sd)

## Get average based on number of sets combined [This if for 3]
superbound_avg$AvgVdW <- format(superbound_avg$VdW / sets, digi
ts=4, format="f")
superbound_avg$AvgVdWSD <- format(superbound_sd$VdW / sets, dig
its=4, format="f")

## If you for some reason care about StdErr, then you'd uncomme
nt this
## Yes, it's weird that StdErr has a SD, but that's just a sani
ty thing
#superbound_avg$AvgStdErr <- format(superbound_avg$StdErr / set
s, digits=4, format="f")
#superbound_avg$AvgStdErrSD <- format(superbound_sd$StdErr / se
ts, digits=4, format="f")
```

```
save_cols_Av <- superbound_avg[,c("ResidueA", "ResidueB", "AvgV
dW", "AvgVdWSD")]
#save_cols_Av <- superbound_avg[,c("ResidueA", "ResidueB", "Avg
VdW", "AvgVdWSD", "AvgStdErr", "AvgStdErrSD")]

only_ROI_rows_Av <- filter(save_cols_Av, ResidueA == ROI | Resi
dueB == ROI)

## Change the NA from standard deviation to 0
only_ROI_rows_Av[(ROI),4] <- 0

## Create a copy of the parsed data to format
clean_rows_Av <- data.frame(only_ROI_rows_Av)

## Pattern searching converted it to a character string, so bac
k to numeric
clean_rows_Av$AvgVdW <- as.numeric(clean_rows_Av$AvgVdW)
clean_rows_Av$AvgVdWSD <- as.numeric(clean_rows_Av$AvgVdWSD)

## Limit to 4 sig figs after decimal
clean_rows_Av$AvgVdW <- formatC(clean_rows_Av$AvgVdW, digit
s=4, format="f")
clean_rows_Av$AvgVdWSD <- formatC(clean_rows_Av$AvgVdWSD, digit
s=4, format="f")

## Set the two residues surrounding the ROI to zero
## This is because energy is overpowering due to other energy t
erms
## So if ROI=100, you remove matches between 99 & 100 as well a
s 100 & 101
if (ROI != 1) {clean_rows_Av[(ROI-1),3] <- 0
clean_rows_Av[(ROI-1),4] <- 0
}
clean_rows_Av[(ROI+1),3] <- 0
clean_rows_Av[(ROI+1),4] <- 0

#-----------------------------------------------------------------
---------#
#-----------------------VDW OUTFILES-----------------------
#
#-----------------------------------------------------------------
---------#

## Now write a tab-delimited outfile!
## Don't care about the index rownames
```

```
#write.table(clean_rows_Av, file = A_vdw, sep="\t", row.names=F
ALSE, quote=FALSE)

## Write a whitespace-delimited outfile!
sink(A_vdw, type=c("output"))
print(clean_rows_Av, row.names=FALSE)
sink()

#---------------------------------------#
#--Create the TOTAL (Coul + vdW) files--#
#---------------------------------------#

## Combine into one dataset
## Use all columns from _Ac and the VdW and VdWSD columns from
_Av
## Note: this makes it a matrix
combine_Acv = cbind(clean_rows_Ac, clean_rows_Av[,3:4])

## Formatting the rows converted it to a character string, so b
ack to numeric again!
combine_Acv$AvgCoulomb <- as.numeric(combine_Acv$AvgCoulomb)
combine_Acv$AvgCoulombSD <- as.numeric(combine_Acv$AvgCoulombS
D)
combine_Acv$AvgVdW <- as.numeric(combine_Acv$AvgVdW)
combine_Acv$AvgVdWSD <- as.numeric(combine_Acv$AvgVdWSD)

## Your data are now ResidueA ResidueB AvgCoul AvgCoulSD AvgVd
W AvgVdWSD
## Append a column called AvgIntTot thats the sum of AvgCoul an
d AvgVdW
combine_Acv$AvgIntTot <- (combine_Acv$AvgCoulomb + combine_Ac
v$AvgVdW)

## Now append a column that's the avg standard deviation
combine_Acv$AvgStdDev <- (combine_Acv$AvgCoulombSD + combine_Ac
v$AvgVdWSD) / 2

## Create a new variable that's just ResidueA ResidueB AvgIntTo
t AvgStdDev
save_cols_tot <- combine_Acv[,c("ResidueA", "ResidueB", "AvgInt
Tot", "AvgStdDev")]

## Sanity Check!
## Set the two residues surrounding the ROI to zero
## This is because energy is overpowering due to other energy t
```

```
erms
## So if ROI=100, you remove matches between 99 & 100 as well a
s 100 & 101
if (ROI != 1) {save_cols_tot[(ROI-1),3] <- 0
save_cols_tot[(ROI-1),4] <- 0
}
save_cols_tot[(ROI+1),3] <- 0
save_cols_tot[(ROI+1),4] <- 0

#-----------------------------------------------------------------
---------#
#-------------------TOTAL INTERACTION OUTFILE
S-------------------#
#-----------------------------------------------------------------
---------#

## Now write a tab-delimited outfile!
## Don't care about the index rownames
#write.table(save_cols_tot, file = A_tot, sep="\t", row.names=F
ALSE, quote=FALSE)

## Write a whitespace-delimited outfile!
sink(A_tot, type=c("output"))
print(save_cols_tot, row.names=FALSE)
sink()
```

EDA Results Analysis with R: Difference of Averaged Systems

The following script can be used to get the difference between two systems that have already been averaged with `rmagic-EDA-avg.r` (page 98).

rmagic-EDA-diffs-sysA-sysB.r

```
## Run this with "Rscript rmagic.r"
## (Assuming you've already installed R...)

#-----------------------------------------------------------------
#
#--Specify the paths to the Files from master-analysis-EDA.sh--
#
#-----------------------------------------------------------------
#

## This script has been pre-built for 2 systems with 3 replicat
es
## More or less than 3 reps (up to 5) can be achieved through
## Commenting or uncommenting

## Paths to the -tot- files
## Set A (system 1)
infileACV <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/WT-System-A/systemA_EDA_tot_avg-clean.txt")

##Set B (system 2)
infileBCV <- Sys.glob("/absolute/path/to/the/analysis/files/fo
r/WT-System-B/systemB_EDA_tot_avg.txt")

#----------------------------#
#--Define your outfile names--#
#----------------------------#

## A - B
TOTAB <- "WT_sysA-sysB_total_interaction_res123_avg.dat"

## This is the residue of interest
X_val <- "123"

#-----------------------------------------------------------------
--------#
#---------Behind the Curtain: No Need to Modify Past This Lin
e---------#
#-----------------------------------------------------------------
--------#

## Use the data tables package to read in data frames
## Remove comment to install locally
#install.packages("data.table")
library(data.table)
```

```
## Use the abind package to combine data frames
## Remove comment to install locally
#install.packages("abind")
library(abind)

## Turn off scientific notation
options(scipen = 999)

#------------------#
#--Begin with COUL--#
#------------------#

## Reading each file as a data.table.
## Bonus - fread is much faster than read.csv
combineACV <- fread(infileACV, header=TRUE)
colnames(combineACV) <- c("R1", "R2", "TotAvg", "TotStdD")

combineBCV <- fread(infileBCV, header=TRUE)
colnames(combineBCV) <- c("R1", "R2", "TotAvg", "TotStdD")

## Redefine as a data frame
combineACV <- as.data.frame(combineACV)

combineBCV <- as.data.frame(combineBCV)

## They're not numbers, so make them numbers
combineACV$TotAvg <- as.numeric(as.character(combineACV$TotAv
g))
combineACV$TotStdD <- as.numeric(as.character(combineACV$TotStd
D))

combineBCV$TotAvg <- as.numeric(as.character(combineBCV$TotAv
g))
combineBCV$TotStdD <- as.numeric(as.character(combineBCV$TotStd
D))

## Combine A res numbers, tot average, tot average, tot stdev,
tot stev
combineTotCV <- abind(combineACV[,1:3], combineBCV[,3], combine
ACV[,4], combineBCV[,4], along=2)

## Rename the columns
colnames(combineTotCV) <- c("R1", "R2", "ATotalE", "BTotalE",
"AAvgStd", "BAvgStd")
```

```
## Redefine as a data frame
combineTotCV <- as.data.frame(combineTotCV)

## If the R1 column doesn't equal X_val, use R1. Else, use R2.
combineTotCV$Residue <- ifelse((combineTotCV$R1 != X_val), as.n
umeric(as.character(combineTotCV$R1)), as.numeric(as.characte
r(combineTotCV$R2)))

## They're not numbers, so make them numbers
combineTotCV$ATotalE <- as.numeric(as.character(combineTotCV$AT
otalE))
combineTotCV$BTotalE <- as.numeric(as.character(combineTotCV$BT
otalE))
combineTotCV$AAvgStd <- as.numeric(as.character(combineTotCV$AA
vgStd))
combineTotCV$BAvgStd <- as.numeric(as.character(combineTotCV$BA
vgStd))

## Multiply B * -1
## THIS WILL DO A - B!!
combineTotCV$BTotalE <- (combineTotCV$BTotalE*(-1.0000000000))
combineTotCV$DiffE <- rowSums(combineTotCV[, c("ATotalE", "BTot
alE")])

## Get the Avg Stdev
combineTotCV$AvgSTDEV <- rowMeans(combineTotCV[,5:6])

## Create a new variable with just Residue, DiffE, and AvgSTDEV
save_cols_total_CV <- combineTotCV[,c("Residue", "DiffE", "AvgS
TDEV")]

## Limit to 8 sig figs after decimal
save_cols_clean_total_CV <- format(save_cols_total_CV, digit
s=8)

## Explicitly remove the two residues matched next to the resid
ue of interest
## This is because it's more than interaction energy (stuff lik
e bond E too)
## (Note: | is the or operator)
#save_cols_clean_total_CV <- save_cols_clean_total_CV[!(save_co
ls_clean_total_CV$Residue == as.numeric(X_val)+1 | #save_cols_c
lean_total_CV$Residue == as.numeric(X_val)-1),]
```

```
#-----------------------------------------------------------------
-------#
#-------------------------TOT OUTFILE
S------------------------------#
#-----------------------------------------------------------------
-------#

## Now write a tab-delimited outfile!
## Don't care about the index rownames because that's the frame
#write.table(save_cols_clean_total_CV, file = TOTAB, sep="\t",
row.names=FALSE, quote=FALSE)

## Write a whitespace-delimited outfile!
sink(TOTAB, type=c("output"))
print(save_cols_clean_total_CV, row.names=FALSE)
sink()
```

Using gnuplot's Multiplot Feature with Standard Deviations

You can use the `multiplot` feature to make stacked graphs. The files generated with R contain information for standard deviation, so it is possible to add error bars as well.

```
reset
dx=1.
n=2
total_box_width_relative=1.
gap_width_relative=0
d_width=(gap_width_relative+total_box_width_relative)*dx/2.
reset

set encoding iso_8859_1
set term pngcairo enhanced color font "Arial,30" size 1500,105
0;
#set term postscript enhanced color font "Arial,24";

## Let's use gray colors, standard for each plot
# "Sys A" lc rgb "gray40"
# "Sys B" lc rgb "gray80"

#### STACKED PLOTS ######

set output "../total_interaction_systemA-systemB_yoff.png";
set tmargin 0
set bmargin 0
set lmargin 1
set rmargin 1
unset xlabel
set ylabel "Energy (kcal/mol)" offset 0,-4

unset arrow 1
unset arrow 2
unset arrow 3
unset label

set multiplot layout 2,1 margins 0.12,0.94,0.15,0.88 spacing
0,0

unset xtics
set yrange[-80:60]
set xrange[0:455]
set ytics ("" -80,-60,-40,-20,0,20,40,60) nomirror

set key autotitle column nobox samplen 1 noenhanced
set style data boxes
set x2tics border nomirror in out ("GS Linker" 334, "" 346,"DN
A" 431, "" 451)
```

```
# System A
## Column 0 is the row number
## Toggle the commenting for the first 3 lines to have standar
d deviations
#plot "/absolute/path/to/the/analysis/files/for/WT-System-A/sys
temA_EDA_tot_avg.txt" u ($0):($3) w points t "System A" lw 4 p
t 0 lc "black", \
#"/absolute/path/to/the/analysis/files/for/WT-System-A/system
A_EDA_tot_avg.txt" u ($0):($3):($3-$4):($3+$4):($3) w candlesti
cks  fs solid 0.15 t "Avg. Std. Dev." lc rgb "gray40", \
#"/absolute/path/to/the/analysis/files/for/WT-System-A/system
A_EDA_tot_avg.txt" u ($0):($3) w points notitle ls 1 pt 0 lc "b
lack" lw 4;
plot "/absolute/path/to/the/analysis/files/for/WT-System-A/syst
emA_EDA_tot_avg.txt" u ($0):($3) w boxes t "System A" lc rgb "g
ray40"

unset ylabel
unset x2tics
set x2tics border nomirror in out ("" 334, "" 346,"" 431, "" 45
1) #"" 334, "" 346,
set xlabel "Residue number"
set xtics ("1130" 0, "1180" 50, "1230" 100, "1280" 150, "1330"
200, "1380" 250, "1430" 300, \
     "1845" 350, "1895" 400) border nomirror out; #"" 334, "" 3
46, "" 431
set ytics (-80, -60,-40,-20,0,20,40, "" 60) nomirror

#System B
## Column 0 is the row number
## Toggle the commenting for the first 3 lines to have standar
d deviations
#plot "//absolute/path/to/the/analysis/files/for/WT-System-B/sy
stemB_EDA_tot_avg.txt" u ($0):($3) w points t "System B" lw 4 p
t 0 lc "black", \
#"/absolute/path/to/the/analysis/files/for/WT-System-B/system
B_EDA_tot_avg.txt" u ($0):($3):($3-$4):($3+$4):($3) w candlesti
cks  fs solid 0.15 t "Avg. Std. Dev." lc rgb "gray40", \
#"/absolute/path/to/the/analysis/files/for/WT-System-B/system
B_EDA_tot_avg.txt" u ($0):($3) w points notitle ls 1 pt 0 lc "b
lack" lw 4;
plot "/absolute/path/to/the/analysis/files/for/WT-System-B/syst
emB_EDA_tot_avg.txt" u ($0):($3) w boxes t "System B" lc rgb "g
ray40"
```

```
unset multiplot
```

Deprecated Python EDA Scripts

These scripts are being left in the guides for historic reasons. R is the much more clear-cut way to process these data.

> ❗ **Warning:** These scripts were written for an Anaconda installation of Python3.6. Using Python2.7 **will not work**.

Historic: EDA Results Analysis with awk & Python for Specific Residues

An awk command (shown in a bash script here) can be used to separate out the information for a specific residue of interest.

The following script pulls information for residues 436 and 444 from the generated Fortran90 output, creating separated files for the Coulomb and van der Waals energies for the individual residues. The number of lines that each created file has is then listed in a color in the Terminal where the script was executed. The script can be run after making it executable with `chmod u+x awk.sh`. Then, to run the script, use `./awk.sh`.

awk.sh

```
#!/bin/bash

#Template C modification
awk '{if ($2 == 436 || $3 == 436) print}' fort.803 > coul-436
awk '{if ($2 == 436 || $3 == 436) print}' fort.806 > vdw-436

#Complementary C modification
awk '{if ($2 == 444 || $3 == 444) print}' fort.803 > coul-444
awk '{if ($2 == 444 || $3 == 444) print}' fort.806 > vdw-444

CCOLOR='\033[1;33m' #Color for Coulomb terms is yellow
VCOLOR='\033[1;36m' #Color for VDW terms in blue
NC='\033[0m'

echo -e "coul-436 has ${CCOLOR}$(wc -l < coul-436)${NC} lines f
or NStep"
echo -e " vdw-436 has ${VCOLOR}$(wc -l < vdw-436)${NC} lines fo
r NStep"
echo -e "coul-444 has ${CCOLOR}$(wc -l < coul-444)${NC} lines f
or NStep"
echo -e " vdw-444 has ${VCOLOR}$(wc -l < vdw-444)${NC} lines fo
r NStep"

##Other color choices using ANSI escape codes:
#COLOR='\033[0;31m' #Red
#COLOR='\033[1;32m' #Light green; bright
#COLOR='\033[1;33m' #Yellow, bright
#COLOR='\033[33m'   #Yellow, standard
#COLOR='\033[0;34m' #Light blue
#COLOR='\033[1;36m' #Cyan, bright
#COLOR='\033[0;36m' #Cyan, standard
```

The following Python script, once edited to reflect the number of lines that `awk.sh` registered, will find the Coulomb and van der Waals energies for the individual residues by summing them together. These sums will then be printed to the Terminal in color. In order for this to run properly on a cluster, you'll need a local installation of Conda . Otherwise, to run the script, use `python coul-vdw.py` .

`coul-vdw.py`

```
## Use with result files from EDA program and awk-Cres-nan.sh t
o extract out individual residues

import math        ## Remind Python that it can do math

## Use the numbers from awk-Cres-nan.sh output here
Nsteps_Ac = 473    ## coul-436
Nsteps_Av = 473    ## vdw-436
Nsteps_Bc = 473    ## coul-444
Nsteps_Bv = 473    ## vdw-444

## Coulomb for Residue A
ifile_Ac = open("coul-436", "r" )   ## Open the input file, cou
l-436, for reading

Esum_Ac = 0                         ## Set the energy sum as 0
to begin with

for i in range(Nsteps_Ac):          ## Loop for values for N ro
ws of data, specified above
  dummy_Ac = ifile_Ac.readline()    ## Reads in line-by-line
  dummy_Ac = dummy_Ac.split()       ## Splits the data into col
umns
  Ei_Ac = float(dummy_Ac[3])        ## Returns floating point n
umber; uses column 4 [start@0]
  Esum_Ac += Ei_Ac                  ## Appends values for Esum

## VDW for Residue A
ifile_Av = open("vdw-436", "r" )    ## Open the input file, vd
w-436, for reading

Esum_Av = 0                         ## Set the energy sum as 0
to begin with

for i in range(Nsteps_Av):          ## Loop for values for N ro
ws of data, specified above
  dummy_Av = ifile_Av.readline()    ## Reads in line-by-line
  dummy_Av = dummy_Av.split()       ## Splits the data into col
umns
  Ei_Av = float(dummy_Av[3])        ## Returns floating point n
umber; uses column 4 [start@0]
  Esum_Av += Ei_Av                  ## Appends values for Esum


## Coulomb for Residue B
```

```
ifile_Bc = open("coul-444", "r")

Esum_Bc = 0

for i in range(Nsteps_Bc):
  dummy_Bc = ifile_Bc.readline()
  dummy_Bc = dummy_Bc.split()
  Ei_Bc = float(dummy_Bc[3])
  Esum_Bc += Ei_Bc

## VDW for Residue B
ifile_Bv = open("vdw-444", "r")

Esum_Bv = 0

for i in range(Nsteps_Bv):
  dummy_Bv = ifile_Bv.readline()
  dummy_Bv = dummy_Bv.split()
  Ei_Bv = float(dummy_Bv[3])
  Esum_Bv += Ei_Bv

## Get fancy with colors
from colorama import Fore
from colorama import Style

## Prints the floating point number for all values.
## Using colorama, you start print command with 'f' for colorat
ion
print(f"Coulomb for RES 436 \t Esum = {Fore.YELLOW}{Style.BRIGH
T} %f {Style.RESET_ALL} kcal/mol" %Esum_Ac)  ## Esum_Ac
print(f"VDW     for RES 436 \t Esum = {Fore.CYAN}{Style.BRIGH
T} %f {Style.RESET_ALL} kcal/mol" %Esum_Av)    ## Esum_Av
print(f"Coulomb for RES 444 \t Esum = {Fore.YELLOW}{Style.BRIGH
T} %f {Style.RESET_ALL} kcal/mol" %Esum_Bc)  ## Esum_Bc
print(f"VDW     for RES 444 \t Esum = {Fore.CYAN}{Style.BRIGH
T} %f {Style.RESET_ALL} kcal/mol" %Esum_Bv)    ## Esum_Bv
```

Historic: EDA Results Analysis: Plotting All Residues

A more elaborate bash/Python script combination can be used to total the sums for every residue and put them into convenient `coul-byres.dat` and `vdw-byres.dat` files.

byres-EDA-generation.sh

```
#!/bin/bash

#Start loop at 1
f=1

## Set up loop; the "-lt number" should be the number of protei
n residues +1
## i.e. here there's 455 residues before solvation WAT/K+
while [ $f -lt 456 ]; do

## Extract out per residue information from the f90 output
## The -v flag allows you to to define a variable in the comman
d
## Here it's the previously defined f of the loop iteration
awk -v f=$f '{if ($2 == f || $3 == f) print}' fort.803 > cou
l-$f.tmp
awk -v f=$f '{if ($2 == f || $3 == f) print}' fort.806 > vd
w-$f.tmp

f=$[$f+1]
done

## Run the python script
python coul-vdw-byres.py

## Remove the temp files created in the awk step
rm *.tmp

## Rename the output from python to include the current directo
ry name
mv coul-byres.txt ${PWD##**/}-coul-byres.dat
mv vdw-byres.txt ${PWD##**/}-vdw-byres.dat
```

`coul-vdw-byres.py`

```
## Use with result files from EDA program and byres-EDA-generat
ion.sh get plottable byres data

import math        ## Remind Python that it can do math

Nres = 455         ## Number of residues for analysis
Nsteps = Nres - 1  ## Number of residues for calculation
name = list(range(1,456))  ## From 1 to (but not including) 456

## Coulomb
ofile_c = open("coul-byres.txt", "w+")
for x in name:                                  ## Specifies th
e number of the input file
  ifile_c = open("coul-{}.tmp".format(x),"r")  ## Annoying synt
ax to get loop with input files
  Esum_c = 0                       ## Start the count at 0

  for i in range(Nsteps):          ## Loop for values for N row
s of data
    dummy_c = ifile_c.readline()  ## Reads in line-by-line
    dummy_c = dummy_c.split()     ## Splits the data in columns

    ## This statement skips residues directly next to each othe
r (ie 1&2, 2&3)
    ## Because they have crazy values and aren't just the Coul/
VDW
    if int(float(dummy_c[1])) == (int(float(dummy_c[2])) - 1):
      continue
    else:
      Ei_c = float(dummy_c[3])      ## Returns the floating poi
nt number; uses column 4 [start@0]
      Esum_c += Ei_c                ## Appends values for Esum

  ifile_c.close()                  ## Close the opened input fil
e to free up memory
  ofile_c.write ('%d \t %f \n' %(x,Esum_c)) ## Write the residu
e's energy
ofile_c.close()  ## Close the generated file to free up memory

## VDW
ofile_v = open("vdw-byres.txt", "w+")
count = 1
for x in name:
  ifile_v = open("vdw-{}.tmp".format(x),"r")
  Esum_v = 0
```

```
  for i in range(Nsteps):
    dummy_v = ifile_v.readline()
    dummy_v = dummy_v.split()
    if int(float(dummy_v[1])) == (int(float(dummy_v[2]))-1):
      continue
    else:
      Ei_v = float(dummy_v[3])
      Esum_v += Ei_v

  ifile_v.close()
  ofile_v.write ('%d \t %f \n' %(x,Esum_v))

ofile_v.close()
```

Once both scripts exist in the folder with the `fort.803` and `fort.806` files, simply perform:

```
$ chmod u+x byres-EDA-generation.sh
$ ./byres-EDA-generation.sh
```

and generated files will be easily plotted using gnuplot (or another plotting utility of your choice).

Historic: Plotting By Residue Data in gnuplot

Gnuplot is a freely available plotting utility.

The following (`EDA\_plot.gnu`) is an example gnuplot script that can be used to generate graphs of the by-residue Coulomb and vdW energies of 4 different systems. This script would be in a directory that contained the directories of the individual systems, which are then accessed individually in the plot command. The number of residues should be changed in the `set xrange [0:500]` line to reflect the number of residues of the protein.

```
set encoding iso_8859_1
set term postscript enhanced color font "Arial,24";


set xlabel "Residue number"
set ylabel "Coulomb Energy (kcal/mol)"
set xrange [0:500]
set key bottom left Left reverse width 2 height 1

set output "coulomb.eps";
plot "system-1/system-1-coul-byres.dat" u 1:2 w boxes t  "Syste
m 1" lw 4, \
"system-2/system-2-coul-byres.dat" u 1:2 w boxes t  "System 2"
lw 4, \
"system-3/system-3-coul-byres.dat" u 1:2 w boxes t "System 3" l
w 4, \
"system-4/system-4-coul-byres.dat" u 1:2 w boxes t "System 4" l
w 4;

set xlabel "Residue number"
set ylabel "vdW Energy (kcal/mol)"
set key top left Left reverse width 2 height 1

set output "vdw.eps";
plot "system-1/system-1-vdw-byres.dat" u 1:2 w boxes t  "Syste
m 1" lw 4, \
"system-2/system-2-vdw-byres.dat" u 1:2 w boxes t  "System 2" l
w 4, \
"system-3/system-3-vdw-byres.dat" u 1:2 w boxes t "System 3" l
w 4, \
"system-4/system-4-vdw-byres.dat" u 1:2 w boxes t "System 4" l
w 4;
\end{lstlisting}

To run the gnuplot script and generate the plots, use:
\begin{lstlisting}[style=P1]
$ gnuplot EDA_plot.gnu
```

Historic: EDA Results Analysis: Plotting the Interactions of a Single Residue

An bash script consisting of a series of awk commands can be used to extract out the matches for a single residue of interest and prepare them for plotting with Gnuplot . This script will go through and set the values for the two adjacent residues to zero, as their interactions are affected by stuff like dihedral angles.

`bashbyres.sh`

```
#!/bin/bash


## Prints just the interactions of 1&f, 2&f....
## Set f as the residue of interest, and this should do the res
t :D
f=436
g=$[f+1]
e=$[f-1]

## First pass at Coulomb interaction
awk -v f=$f '{if ($2 == f) printf "%8s %5s %22s %22s\n", $1,
$3, $4, $5}' fort.803 > coul-$f-byres.tmp
awk -v f=$f '{if ($3 == f) printf "%8s %5s %22s %22s\n", $1,
$2, $4, $5}' fort.803 >> coul-$f-byres.tmp

## First pass at vdW interaction
awk -v f=$f '{if ($2 == f) printf "%8s %5s %22s %22s\n", $1,
$3, $4, $5}' fort.806 > vdw-$f-byres.tmp
awk -v f=$f '{if ($3 == f) printf "%8s %5s %22s %22s\n", $1,
$2, $4, $5}' fort.806 >> vdw-$f-byres.tmp


## Coulomb interaction

## Setting the adjacent residues to zero because they aren't ju
st C/vdW
## Adjust the column with value for both adjacents
awk -v var1=$g -v f=$f '$2==var1{$3="0"} {printf "%8s %5s %22s
%22s\n", $1, $2, $3, $4}' coul-$f-byres.tmp > coul-$f-byres.tmp
2

mv coul-$f-byres.tmp2 coul-$f-byres.tmp

awk -v var2=$e -v f=$f '$2==var2{$3="0"} {printf "%8s %5s %22s
%22s\n", $1, $2, $3, $4}' coul-$f-byres.tmp > coul-$f-byres.tmp
2

mv coul-$f-byres.tmp2 coul-$f-byres.tmp

## Adjust the column with standard energy for both adjacents

awk -v var1=$g -v f=$f '$2==var1{$4="0"} {printf "%8s %5s %22s
%22s\n", $1, $2, $3, $4}' coul-$f-byres.tmp > coul-$f-byres.tmp
2
```

```
mv coul-$f-byres.tmp2 coul-$f-byres.tmp

awk -v var2=$e -v f=$f '$2==var2{$4="0"} {printf "%8s %5s %22s
%22s\n", $1, $2, $3, $4}' coul-$f-byres.tmp > coul-$f-byres.tmp
2

mv coul-$f-byres.tmp2 coul-$f-byres.tmp && mv coul-$f-byres.tm
p coul-$f-byres

## Sort by the value numerically so gnuplot doesn't get interes
ting...
sort -k2,4n coul-$f-byres > coul-$f-byres.tmp
mv coul-$f-byres.tmp ${PWD##**/}-coul-$f-res.dat

## vdW interaction

## Setting the adjacent residues to zero because they aren't ju
st C/vdW
awk -v var1=$g -v f=$f '$2==var1{$3="0"} {printf "%8s %5s %22s
%22s\n", $1, $2, $3, $4}' vdw-$f-byres.tmp > vdw-$f-byres.tmp2

mv vdw-$f-byres.tmp2 vdw-$f-byres.tmp

awk -v var2=$e -v f=$f '$2==var2{$3="0"} {printf "%8s %5s %22s
%22s\n", $1, $2, $3, $4}' vdw-$f-byres.tmp > vdw-$f-byres.tmp2

mv vdw-$f-byres.tmp2 vdw-$f-byres.tmp

awk -v var1=$g -v f=$f '$2==var1{$4="0"} {printf "%8s %5s %22s
%22s\n", $1, $2, $3, $4}' vdw-$f-byres.tmp > vdw-$f-byres.tmp2

mv vdw-$f-byres.tmp2 vdw-$f-byres.tmp

awk -v var2=$e -v f=$f '$2==var2{$4="0"} {printf "%8s %5s %22s
%22s\n", $1, $2, $3, $4}' vdw-$f-byres.tmp > vdw-$f-byres.tmp2

mv vdw-$f-byres.tmp2 vdw-$f-byres.tmp && mv vdw-$f-byres.tmp vd
w-$f-byres

sort -k2,4n vdw-$f-byres > vdw-$f-byres.tmp
mv vdw-$f-byres.tmp ${PWD##**/}-vdw-$f-res.dat
```

Once the script exist in the folder with the `fort.803` and `fort.806` files, simply perform:

```
$ chmod u+x bashbyres.sh
$ ./bashbyres.sh
```

and generated files will be easily plotted using gnuplot (or another plotting utility of your choice).

Historic: Plotting A Single Residue's Data in gnuplot

Gnuplot is a freely available plotting utility.

The following (`EDA_plot.gnu`) is an example gnuplot script that can be used to generate graphs of the by-residue Coulomb and vdW energies for a single residue in the system. This script would be in a directory that contained the directories of the individual systems, which are then accessed individually in the plot command. The number of residues should be changed in the `set xrange [0:500]` line to reflect the number of residues of the protein.

```
set encoding iso_8859_1
set term postscript enhanced color font "Arial,24";

set xlabel "Residue number"
set ylabel "Coulomb Energy (kcal/mol)"
set xrange [0:500]
set key bottom left Left reverse width 2 height 1

#For points use "w points" instead of "w boxes" (boxplots)
#For lines use "w lines"

set title "Coulomb Interactions for Residue X"
set output "coulomb-436-system1.eps";
plot "system1/system1-coul-436-res.dat" u 2:($3) w boxes t "Sys
tem1" lw 4;

set xlabel "Residue number"
set ylabel "vdW Energy (kcal/mol)"
set key top left Left reverse width 2 height 1

set title "van der Waals Interactions for Residue X"
set output "vdw-436-system1.eps";
plot "system-1/system-1-vdw-436-res.dat" u 2:($3) w boxes t "Sy
stem 1" lw 4;
```

To run the gnuplot script and generate the plots, use:

```
$ gnuplot EDA_plot.gnu
```

Historic: Plotting a Difference Between 2 Systems at a Single Residue in gnuplot

The following (`EDA_plot_subtraction.gnu`) is an example gnuplot script that can be used to generate graphs of the by-residue Coulomb and vdW energies for the difference between two systems at a single residue. The example residue here is `436`.

Before this script is run, you'll need to use the `paste` command to combine data together into columns. The data that should be combined comes from `bashbyres.sh` (see above (page 129)).

For the Coulomb data, use something like:

```
$ paste WT/resid436/resid436-coul-436-res.dat MUT-A/resid436/re
sid436-coul-436-res.dat MUT-B/resid436/resid436-coul-436-res.da
t MUT-C/resid436/resid436-coul-436-res.dat > combodata-coul-WT-
res436.dat
```

For the vdW data, use something like:

```
$ paste WT/resid436/resid436-vdw-436-res.dat MUT-A/resid436/res
id436-vdw-436-res.dat MUT-B/resid436/resid436-vdw-436-res.dat M
UT-C/resid436/resid436-vdw-436-res.dat > combodata-vdw-WT-res43
6.dat
```

```
#paste WT/resid436/resid436-coul-436-res.dat MUT-A/resid436/res
id436-coul-436-res.dat MUT-B/resid436/resid436-coul-436-res.da
t MUT-C/resid436/resid436-coul-436-res.dat > combodata-coul-WT-
res436.dat

#paste WT/resid436/resid436-vdw-436-res.dat MUT-A/resid436/resi
d436-vdw-436-res.dat MUT-B/resid436/resid436-vdw-436-res.dat MU
T-C/resid436/resid436-vdw-436-res.dat > combodata-vdw-WT-res43
6.dat

set encoding iso_8859_1
set term postscript enhanced color font "Arial,24";

set xlabel "Residue number"
set ylabel "Coulomb Energy (kcal/mol)"
set xrange [0:455]

set key bottom left Left reverse width 2 height 1

## WT - MUT A  ##

set output "coulomb-WT-MUTA-res436.eps";
plot "combodata-coul-WT-res436.dat" u 2:($3-$7) w boxes t  "WT
- MUT A" lw 4;

set xlabel "Residue number"
set ylabel "vdW Energy (kcal/mol)"
set key top left Left reverse width 2 height 1

set output "vdw-WT-MUTA-res436.eps";
plot "combodata-vdw-WT-res436.dat" u 2:($3-$7) w boxes t  "WT
- MUT A" lw 4;

## WT - MUT B ##
set xlabel "Residue number"
set ylabel "Coulomb Energy (kcal/mol)"

set output "coulomb-WT-MUTB-res436.eps";
plot "combodata-coul-WT-res436.dat" u 2:($3-$11) w boxes t  "W
T - MUT B" lw 4;

set xlabel "Residue number"
set ylabel "vdW Energy (kcal/mol)"
set key top left Left reverse width 2 height 1
```

```
set output "vdw-WT-MUTB-res436.eps";
plot "combodata-vdw-WT-res436.dat" u 2:($3-$11) w boxes t  "WT
- MUT B" lw 4;

## WT - MUT C ##
set xlabel "Residue number"
set ylabel "Coulomb Energy (kcal/mol)"

set output "coulomb-WT-MUTC-res436.eps";
plot "combodata-coul-WT-res436.dat" u 2:($3-$15) w boxes t  "W
T - MUT C" lw 4;

set xlabel "Residue number"
set ylabel "vdW Energy (kcal/mol)"
set key top left Left reverse width 2 height 1

set output "vdw-WT-MUTC-res436.eps";
plot "combodata-vdw-WT-res436.dat" u 2:($3-$15) w boxes t  "WT
- MUT C" lw 4;
```

Historic: Plotting Side-by-Side Datasets in gnuplot

You may want to plot the energies of two systems side-by-side. This can get really messy, since your number of residues is then doubled. It takes a math-based work around to force gnuplot to do this, so an example script has been included here.

```
reset
dx=1.
n=2
total_box_width_relative=1.
gap_width_relative=0
d_width=(gap_width_relative+total_box_width_relative)*dx/2.
reset

set encoding iso_8859_1
set term pngcairo enhanced color font "Arial,30" size 1500,105
0;

set output "coulomb_1-2.png";

set xlabel "Residue number"
set ylabel "Coulomb Energy (kcal/mol)"
set xrange [0:455]

set boxwidth total_box_width_relative/n relative
set style fill solid 0.8 noborder

set xtics ("1130" 0, "1180" 50, "1230" 100, "1280" 150, "1330"
200, "1380" 250, "1430" 300, \
     "1845" 350, "1895" 400) border nomirror out; #"" 334, "" 3
46, "" 431
set x2tics border nomirror out rotate by 15 ("DNA" 431, "" 45
1) #

set arrow 1 from first 334,-39 to first 346,-39 lw 2 nohead
set arrow 2 from first 334,-40 to first 334,-38 lw 1 nohead
set arrow 3 from first 346,-40 to first 346,-38 lw 1 nohead
set label "Linker" at 310,-44 font ",20"

set key top right Right width 2 height 1 font ",20"

#System One
#System Two
plot "system-one-avg_EDA_coul-clean-nocolon.txt" u ($1):2 w box
es t "System One" lc rgb "#0000FA", \
"system-two-avg_EDA_coul-clean-nocolon.txt" u ($1+d_width):2 w
boxes t "System Two" lc rgb "#FA7D00";
```

Historic: Plotting Side-by-Side Datasets in matplotlib

Sometimes people don't like the side-by-side variant made using gnuplot. You then spend several hours trying to make them have thicker lines or what have you by using a different plotting program. Now there's a script for a barchart with two datasets generated using matplotlib .

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import statsmodels.api as sm
import pandas as pd
from tables import *

x1, y1 = np.loadtxt("../system-one-avg_EDA_total.txt",unpack=Tr
ue,delimiter='\t')
x2, y2 = np.loadtxt("../system-two-avg_EDA_total.txt",unpack=Tr
ue,delimiter='\t')

plt.rcParams.update({'font.size': 24})
plt.rcParams.update({'figure.autolayout': True})

labelsx2 = [1130, 1180, 1230, 1280, 1330, 1380, 1430, 'Linke
r', '', 1895, '       DNA']
placesx2 = [0, 50, 100, 150, 200, 250, 300, 333, 347, 400, 430]

n_residues = 455
index = np.arange(n_residues)
bar_width = 0.5

fig = plt.gcf()
#fig.set_size_inches(11,8.5)
fig.set_size_inches(22,17)

#111 = 1 row x 1 column x 1 index
ax = plt.subplot(111)
ax.axes.get_xaxis()
ax.set_xticks(index + bar_width / 2)
ax.set_xticks(placesx2)
ax.set_xticklabels(labelsx2, fontdict=None, minor=False)
ax.set_xlim(0,456)
ax.tick_params(axis='both', which='major', pad=10, length=10)
systone = ax.bar(index, y1, bar_width, color='#0000FA',align='c
enter',label='system one')
systtwo = ax.bar(index + bar_width, y2, bar_width, color='#FA7D
00',align='center',label='system two')
plt.ylabel('Energy (kcal/mol)')
plt.xlabel('Residue Number')
plt.legend(handles=[systone,systtwo])
plt.savefig('system_total_barchart.png')
plt.close('system_total_barchart.png')
```

Note: You'll want to change `nresidues` and the `labelx2` and `placesx2` categories to reflect your system. The labels specified for `systone` and `systtwo` (which are dummy variables, you can make them whatever you want, really) are what appear in the graph legend. The two specified colors, `#0000FA` and `#FA7D00`, are blue and orange, respectively.

Historic: Plotting Multiple Datasets using gnuplot's Multiplot Feature

You can use the multiplot feature to make stacked graphs.

```
set encoding iso_8859_1
#set term postscript enhanced color font "Arial,24";
set term pngcairo enhanced color font "Arial,30" size 1500,105
0;

#### STACKED PLOTS ######

set output "coulomb_stacked_yoffset.png";
set tmargin 0
set bmargin 0
set lmargin 1
set rmargin 1
unset xlabel
set ylabel "Coulomb Energy (kcal/mol)" offset 0,-4

unset arrow 1
unset arrow 2
unset arrow 3
unset label

#Ask for 3, use 2
#set multiplot layout 3,1 margins 0.05,0.95,.1,.99 spacing 0,0
set multiplot layout 2,1 margins 0.12,0.94,0.15,0.88 spacing
0,0
#set multiplot layout 2,1 margins 0.12,0.88,0.15,0.85 spacing
0,0

unset xtics
set yrange[-80:60]
set ytics ("" -80,-60,-40,-20,0,20,40,60) nomirror

set key autotitle column nobox samplen 1 noenhanced
set style data boxes
set x2tics border nomirror in out ("GS Linker" 334, "" 346,"DN
A" 431, "" 451)

#System One
plot "system-one-avg_EDA_coul-clean-nocolon.txt" u ($1):2 w box
es t "System One" lc "medium-blue"

unset ylabel
unset x2tics
set x2tics border nomirror in out ("" 334, "" 346,"" 431, "" 45
1) #"" 334, "" 346,
set xlabel "Residue number"
```

```
set xtics ("1130" 0, "1180" 50, "1230" 100, "1280" 150, "1330"
200, "1380" 250, "1430" 300, \
     "1845" 350, "1895" 400) border nomirror out;
set ytics (-80, -60,-40,-20,0,20,40, "" 60) nomirror

#System Two
plot "system-two-avg_EDA_coul-clean-nocolon.txt" u ($1):($2) w
boxes t "System Two" lc "dark-orange"

unset multiplot
```

Mapping Data to Structures

UCSF Chimera is a free molecular modeling program (available for Unix, MacOS X, and Windows) with a lot of tools and integrated resources for biological systems. Chimera has already been installed on the lab computers, but you can install it on your personal computer as well.

Unlike VMD (page 56), Chimera does not require flags to load structures from the command line. That said, of all the file formats AMBER uses (i.e. PDB, mdcrd, prmtop, inpcrd, rst, etc.) Chimera can only open a PDB file, so it is crucial to use VMD to save a PDB of the last frame of a simulation for the purpose of generating images.

There are two likely types of difference graphs that you'll want to make images for based on residue, and those are matrix correlations (page 148) and EDA plots by residue (page 0). The first step for both of these is to create a .txt file with certain information for defining an `attribute` in Chimera. To do this, the first 3-5 lines of the data file set up the type of plotting that will be created, following an *identifier: value* fashion.

1. It is good housekeeping to start this file with a comment describing what the data are. This might just be the system names that the difference was drawn from.
    - Ex: `#KINASE-WT--MUT-A`
2. This line is the attribute name that Chimera will use to reference the data. It cannot begin with a number, underscore, or capital letter. That said, it can contain underscores and alphanumeric characters, as long as no spaces are used.
    - Ex: `attribute: coul-EDA-WT--MUT-A`
3. This line specifies the match mode. By default, it is set to `any`, but other options include `non-zero` and `1-to-1`.
    - Ex: `match mode: 1-to-1`
4. This line specifies the recipient of the matching as `atoms` (default), `residues`, or `molecules`.
5. Finally, if your data has values set as `none`, then you can choose whether to treat those as they Python value None, as a string, or delete them. You shouldn't need this, but if you ever find yourself in that situation, just know it exists.
    - Ex: `recipient: residues`

The remaining lines are formatted as `[Tab] :atomnumber [Tab] attribute-value`. So your data would appear left-aligned like:

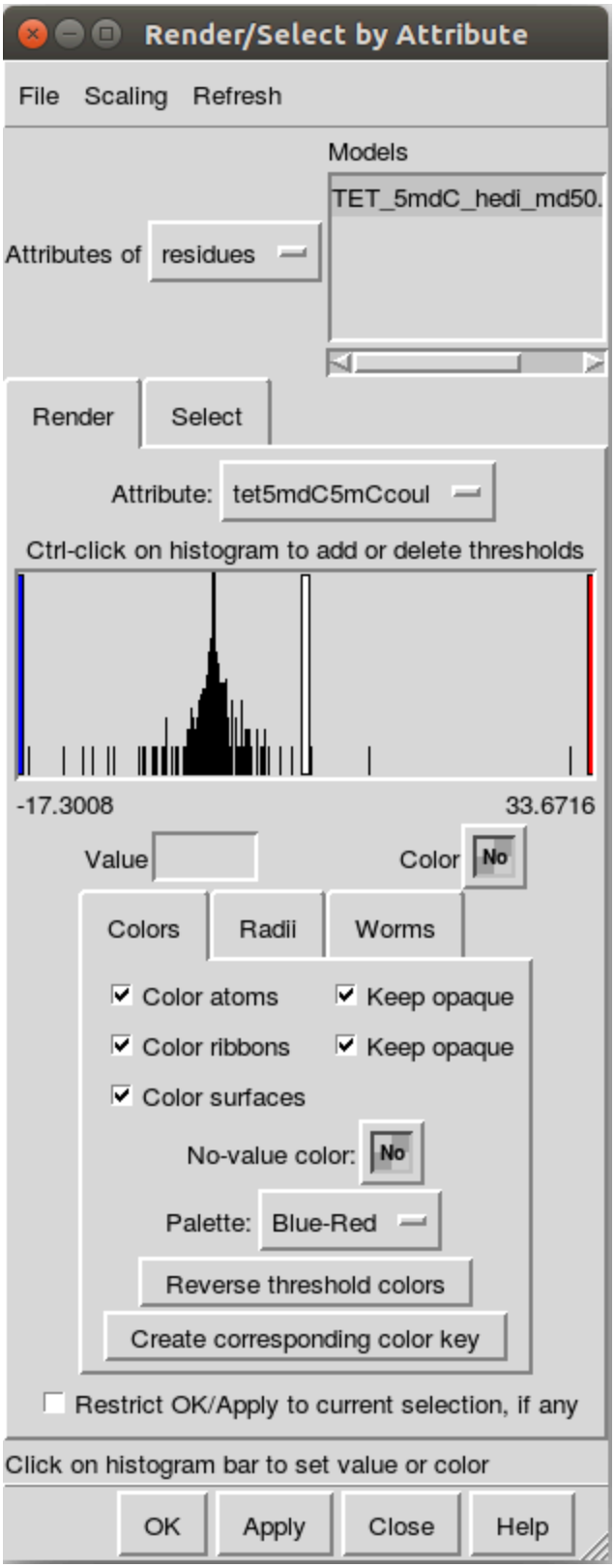
```
        :1   1.325
        :2   -0.313
        :3   -2.109
```

A sample bash script for making these files is `chimeraprint.sh` (page 151).

There's more specific information about generating these attribute lists based on matrix correlation data (page 148) and EDA data (page 150).

To actually create the graphs, open the PDB saved from the final frame in Chimera. Then, follow `Tools → Structure Analysis → Define Attribute` in Chimera's main menu. This will open a list of files and folders. Select the data file (that has the attribute information just described) for what you're trying to plot, keeping `Open/Render/Select by Attribute` checked. If you're making several images at a time, you may also want to check the `Keep dialog up after Open` box.

Once the attribute has been loaded in, a box with setting information appears (Fig. \ref{fig:redbluegraphs}).



The Render/Select by Attribute Chimera menu.

The steps for making images include:

1. Clicking on `Reverse Threshold Colors`, which will make any negative values red and positive values blue.
2. Selecting the red bar (click in the histogram area) and manually setting the value to
    - `-0.5` for matrix correlations
    - `-1` or `-2` for EDA.

    by putting it in the `Value` box and hitting the enter key.
3. Selecting the blue bar (click in the histogram area) and manually setting the value to
    - `0.5` for matrix correlations
    - `1` or `2` for EDA, being consistent with the red bar.

    by putting it in the `Value` box and hitting the enter key.
4. Selecting the white bar (click in the histogram area) and manually set the color to a nice yellow color, with some sort of opacity. For me, that color is near `#ffffffff9792` (Chimera may change it on you, it's okay, just keep it consistent) with an A value (which is the opacity) of `0.591`.
5. Hit `Apply`
6. Select any residues that you wanted to display by using Chimera's command line `Favorites → Command Line` and typing something like

    ```
    select :200
    ```

    where `200` would be the residue number. Then, follow `Actions → Atoms/Bonds → Show` to show the residue. To deselect the residue (i.e. make it not have a green outline), either do `deselect :200` in Chimera's command line or hit the CTRL key and click in an area of white space.
7. Orient the protein in the manner that you would like it to be photographed.
8. Go to `File → Save Image` in the menu bar. The settings for this should be a PNG format with `4x4 supersample` and `transparent background`. Like in VMD, if you have multiple structures loaded in, specifically highlight the one you're interested in.
9. Once saved, do a happy dance! You created yet another image! Incredible!

If you need to create a key for the coloration that you just did, see the section on Color Keys (page 152).

Matrix Correlation Information

Once the *cpptraj* matrix correlation data has been processed with the matrix correlation Python script (page 38), a list of values corresponding to each line in the protein is generated. That file is literally just lines of numbers, and is not ready for use in Chimera attribute mapping without some finagling. Behold: `matcor-chimera-numbers.sh`. Instead of individually pasting numbers and adding attribute lines multiple times over for every system you wish to plot, you can change a few lines in the script and generate the numeric lists a little easier.

matcor-chimera-numbers.sh

```
#!/bin/bash

## Define variables
## f will name your files, RESA is the first residue name, RES
B is the
## second residue name, and INFILE is the file created when mak
ing the
## correlation plots when using the Python script
f="SYSTEM"
RESA="WT"
RESB="MUTA"
INFILE='/path/to/file/created/by/matr_corr.py'

## Find number of lines in generated matrix value (i.e. number
of residues)
## And paste those numbers and data together into one file
a=$(wc $INFILE)
LC=$(echo $a|cut -d' ' -f1)
seq 1 $LC > ${f}-numbers.txt
paste ${f}-numbers.txt $INFILE > ${f}-pastenumbers.txt

## Set up information for Chimera attribues
## Note: attribute cannot start with a capital letter
echo "#SYSTEM${RESA}SYSTEM${RESB}" > ${f}_mc_chimera.txt
echo "attribute: system${RESA}${RESB}mc" >> ${f}_mc_chimera.txt
echo "match mode: 1-to-1" >> ${f}_mc_chimera.txt
echo "recipient: residues" >> ${f}_mc_chimera.txt

## This will print the formatted residue number and value
awk '{printf "\t:%-3s\t%-5s\n", $1, $2}' ${f}-pastenumbers.txt
>> ${f}_mc_chimera.txt

## Remove the lists of numbers generated with paste
rm ${f}-numbers.txt
rm ${f}-pastenumbers.txt
```

EDA Plots by Residue

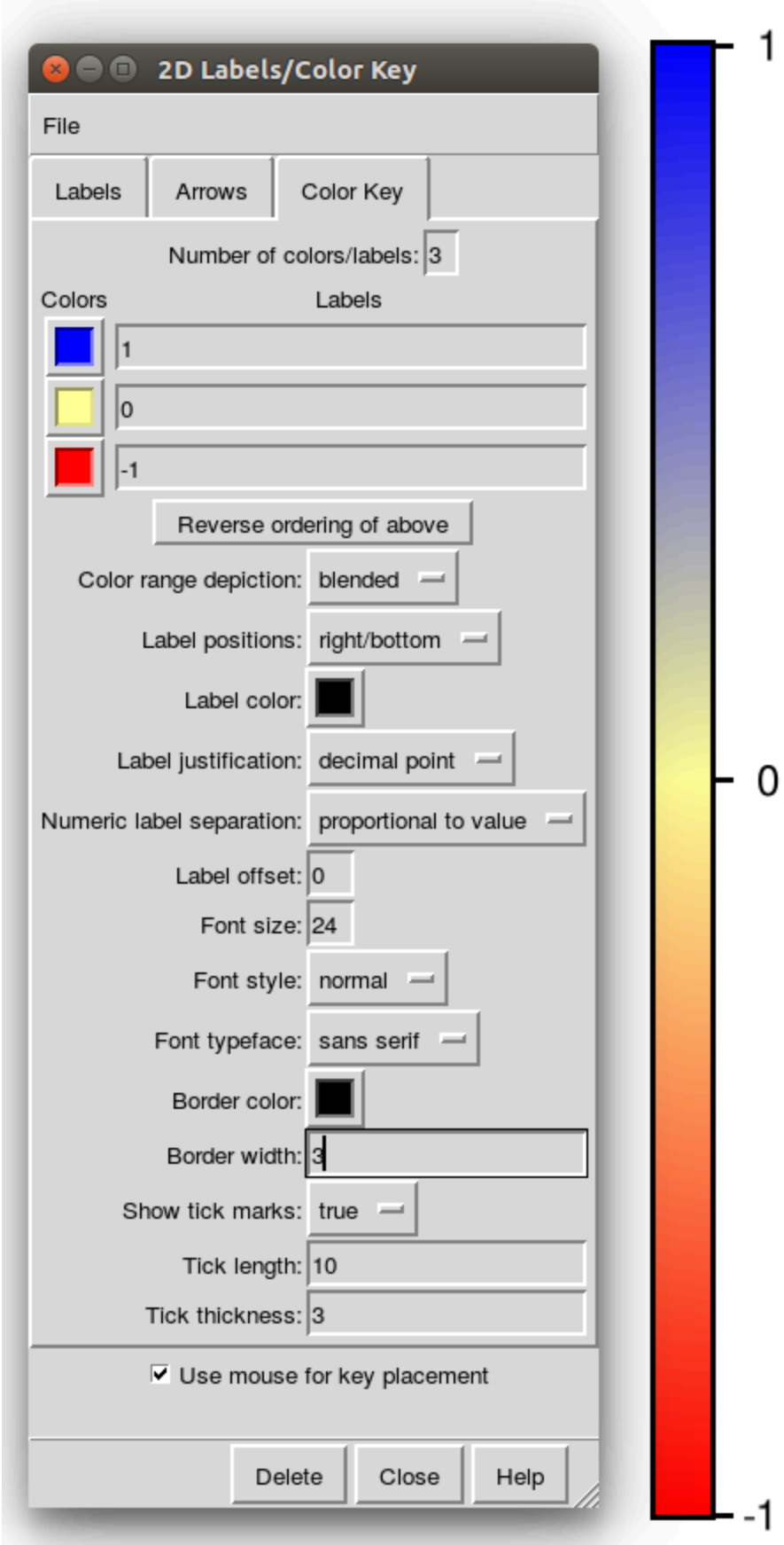
In an effort to quickly generate the EDA plots based on data that has already been used in previous steps, `chimeraprint.sh` was written. This script generates the files needed for difference images for by-residue data that has been processed using `bashbyres.sh` (page 0) and using the associated paste (page 0) command to create a combined data file.

chimeraprint.sh

```
#!/bin/bash

## Define variables
## f will name your files, RESA is the first residue name, RES
B is the second residue name
## g and h are dependent on the columns that the energy was fou
nd in the combined data used
## after doing "paste" on bashbyres.sh data files
f="SYSTEM-WT--MUT-A"
g=3 #Column with WT energy
h=7 #Column with MUT A energy
RESA="WT"
RESB="MUT-A"

## Coul
## Set up the information for Chimera
## Note: attribute cannot start with a capital letter
echo "#SYSTEM${RESA}SYSTEM${RESB}" > ${f}_EDA_coul_chimera.txt
echo "attribute: system${RESA}${RESB}coul" >> ${f}_EDA_coul_chi
mera.txt
echo "match mode: 1-to-1" >> ${f}_EDA_coul_chimera.txt
echo "recipient: residues" >> ${f}_EDA_coul_chimera.txt

## This will print the residue number ($2) and the difference i
n energy for $g and $h (specified above)
awk -v g=$g -v h=$h '{printf "\t:%-3s\t%-22s\n", $2, ($g -
$h)}' combodata-coul-WT-res436.dat >> ${f}_EDA_coul_chimera.txt

## VDW
echo "#SYSTEM${RESA}SYSTEM${RESB}" > ${f}_EDA_vdw_chimera.txt
echo "attribute: system${RESA}${RESB}vdw" >> ${f}_EDA_vdw_chime
ra.txt
echo "match mode: 1-to-1" >> ${f}_EDA_vdw_chimera.txt
echo "recipient: residues" >> ${f}_EDA_vdw_chimera.txt

awk -v g=$g -v h=$h '{printf "\t:%-3s\t%-22s\n", $2, ($g -
$h)}' combodata-vdw-WT-res436.dat >> ${f}_EDA_vdw_chimera.txt
```

Color Keys

Chimera allows users to develop a color key that matches the scale used in attribute shading. To add a color key to your image (or just make a color key for a previously generated image), follow either `Tools → Depiction → Color Key` or `Tools → Utilities → Color Key`. Both are in newer editions of Chimera, but older editions only have the latter. This path will bring up the Color Key menu (see the image below).


2D Labels/Color Key
File
Labels
Arrows
Color Key
Number of colors/labels: 3
Colors
Labels
1
0
-1
Reverse ordering of above
Color range depiction: blended
Label positions: right/bottom
Label color:
Label justification: decimal point
Numeric label separation: proportional to value
Label offset: 0
Font size: 24
Font style: normal
Font typeface: sans serif
Border color:
Border width: 3
Show tick marks: true
Tick length: 10
Tick thickness: 3
Use mouse for key placement
Delete
Close
Help
1
0
-1

Chimera's Color Key menu, with options for generating a color key, and an example shown from the options given.

The number of colors shown in the scale can be changed in the `Number of colors/labels` box. Clicking on an individual color will bring up the colors menu. The colors depicted in the figure have the Tk color codes of:

- Blue: `#0000ffffffff`
- Yellow: `#ffffffff9792`
- Red: `#ffffffff0000` The key can be blocks of the given colors, or made blended by changing the `Color range depiction` box, and the figure labels can include plain text (i.e. unformatted) unit labels.

Once the settings have been chosen, simply draw the key wherever you would like it. If you click anywhere outside the center of the generated key while the color key box is open, then it will start to redraw the key. Grabbing the center of the drawn key will allow it to be repositioned.

Gnuplot Overview

You've seen the phrase "Gnuplot is a freely available plotting utility" several times if you didn't just skip directly to this section. While Gnuplot is indeed a freely available plotting utility, it is also a command-line driven tool. It can be used to generate publishable graphs and charts with minimal effort. Sure, it takes some time to figure out the syntax, and yes you'll probably make the same graph 20 time to make it "just so," but would you rather take that information from one project and edit a script for the next, or reinvent the wheel each time using Excel? You'll take the free command-line tool? Thought so.

Gnuplot's documentation is very thorough. Information for v5 can be found here .

Custom Settings at Start-Up

A file titled `.gnuplot` can be created in your home directory (`/home/username`) with settings for Gnuplot to use at startup. If you're running on a non-Unix-like system (*cough* Windows), then this file should be titled `GNUPLOT.INI` .

Gnuplot has 8 preset line types that it cycles through when generating graphs. These 8 preset lines are made using colors that are shown to work well for colorblind people. If you regularly make plots with more than 8 lines, you may wish to define new line types in the `.gnuplot` file.

An example `.gnuplot` file, based on the default but with two additional lines. Line 9 is salmon-colored and line 10 is lightslategray.

```
set linetype 1 lc rgb "dark-violet" lw 2 pt 0
set linetype 2 lc rgb "sea-green" lw 2 pt 7
set linetype 3 lc rgb "cyan" lw 2 pt 6 pi -1
set linetype 4 lc rgb "dark-red" lw 2 pt 5 pi -1
set linetype 5 lc rgb "blue" lw 2 pt 8
set linetype 6 lc rgb "dark-orange" lw 2 pt 3
set linetype 7 lc rgb "black" lw 2 pt 11
set linetype 8 lc rgb "goldenrod" lw 2
set linetype 9 lc rgb "#FA8072" lw 4 pt 1
set linetype 10 lc rgb "#778899" lw 4 pt 1
set linetype cycle 10
```

The first 8 colors in the default (and this example) are set based upon colors that are readable for people with different types of color blindness. Some good articles on picking contrasting colors can be found on the somersault18:24 blog and on J*Fly .

Gnuplot Help

Gnuplot has a very in-depth help feature when run interactively. To invoke an interactive run, just type

```
$ gnuplot
```

in the Terminal.

Some useful help commands include:

```
gnuplot> help
Pulls up the help page for the help command.
gnuplot> show colornames
A list of all 111 defined colors and their associated HEX and r
gb codes are available
gnuplot> help linetype
This shows the defaults linetypes for users without a .gnuplot
file
```

Input Information

Gnuplot can take an input with a `.gnu` extension. To run a gnuplot script, do:

```
$ gnuplot name-of-input.gnu
```

The first encountered error, if any, will be printed to the Terminal. The script stops running with the first error, which is why subsequent errors are not reported.

In gnuplot, you need to explicitly set the font encoding to allow for special characters and symbols, such as Greek letters. The encoding line that has been used for everything in this guide is the ISO Latin 1 encoding, with the input command:

```
set encoding iso_8859_1
```

While there are lists available on the Internet , some common symbols for graphs are included in the table below (page 158).

Table: Some symbol codes for ISO encoding

Symbol Name	Symbol	Gnuplot Symbol Code
Angstrom	Å	`{\305}`
Degree	°	`{\260}`
Greek alpha	α	`{/Symbol a}`
Greek chi	χ	`{/Symbol c}`
Greek zeta	ζ	`{/Symbol z}`

Commands in gnuplot can continue across lines when `, \` is placed at the line end. An argument is ended by using a return, or with the semicolon when following lines that ended with `, \`.

Comments are the same as bash shells and Python, where they're initiated with `#`. Comments don't work very well if you're commenting in the middle of a plot command, and it is thus advised that any plot lines you want to comment out are moved to the place after the semicolon.

Multiple graphs can be created with one input, as seen earlier (page 7).

Setting the Terminal Type

There are two very helpful file types to generate gnuplot images with. These are *postscript* and *pngcairo*. Postscript is used with the *.eps* extension and is useful in that it requires minimal formatting–everything is typically scaled appropriately. The downside of using postscript is that the images are written in a different formatting language, and they will need to be converted manually to a *.png* or *.pdf* using commands such as:

```
$ for file in *.eps; do convert $file ${file%.*}-eps-converted-
to.pdf; done
$ for file in *.eps; do convert $file -rotate 90 ${file%.*}.pn
g; done
```

The postscript file format also doesn't allow for image transparency.

Using pngcairo (which will likely require an administrator to run `sudo apt install libcairo2-dev`) means that the image won't be scaled right off, but the resolution will likely be better than a converted `.eps`. Pngcairo also allows for image transparency and more advanced coloration features.

Setting the image type is known as setting the output terminal. For postscript, a line like:

```
set term postscript enhanced color font "Arial,24";
```

would set the terminal using an enhanced postscript format (enhanced allows for better coloration and more difficult line types) with 24 pt Arial font.

For pngcairo, a line like:

```
set term pngcairo enhanced color font "Arial,24" size 750,525;
```

would set the terminal using an enhanced png format (enhanced allows for better coloration and more difficult line types) with 24 pt Arial font at a size of 750 x 525 pixels (corresponding to 5 x 3 inches).

Labels, Keys, and Arrows

Axis labels and keys are set with rather straight-forward command types. Each of the label types in gnuplot also allow for offsetting, which can be especially helpful for keys. Some examples include:

```
set xlabel "Shift Degrees ({\260})"
set ylabel "Slide Degrees ({\260})"
set key top left font "Arial,20" width -1 height 1
set key outside right Left reverse width 2 height 1 font "Aria
l,18" maxrows 3
```

The individual argument structure can be found in the documentation , or by using the `help` (page 157) command interactively.

The tic mark labels can also be forcefully set. This trick is helpful for changing PDB residue numbers to match the actual protein residue numbers. You can use a mix of both the number and words in the tic marks, too. The bottom x labels are `xtics` and the top x labels are `x2tics`.

```
set xtics ("Initial" 0, 5, 10, "Next" 20, 45, "Hour" 60, "End"
80) border nomirror out;
set x2tics border nomirror out rotate by 15 ("Heat" 45)
set label 1 "Stir" at 15,25 font ",18"
```

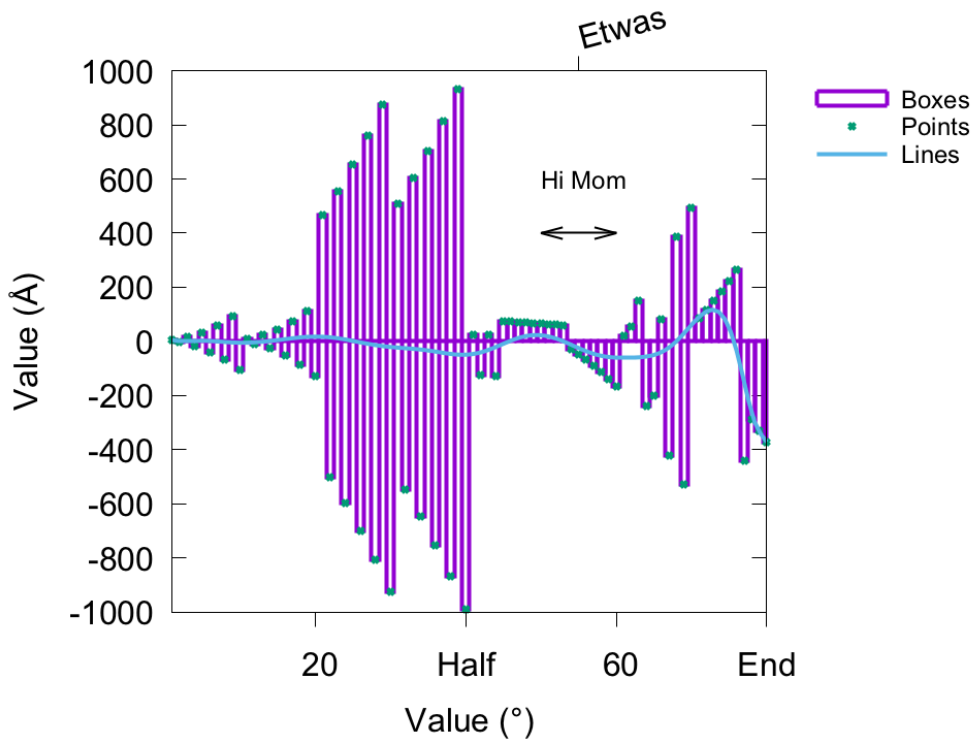
You can place arrows on your graph at set positions. The important word there is set–if they're not positioned in a place that will show up during autoscaling, they will not be visible. The default arrow includes one arrowhead. That can be changed to none or both with `nohead` or `heads`, respectively.

```
set arrow 1 from first 16,54.25 to first 17,60.00 lc rgb "#0085
3E"
set arrow 2 from first 1,65.25 to first 1,60.25 lw 1.2 lc rgb
"#00853E"
set arrow 3 from first 334,10 to first 346,10 lw 2 nohead
set arrow 4 from first 334,9 to first 334,11 lw 1 heads
```

Once labels are unneeded or unwanted, you can use commands like `unset key` and `unset arrow 1` to remove them from the plotting area.

Different Graph Styles

There are multiple plot styles afforded by gnuplot. Some examples include plotting with boxes (useful for EDA plots; `w boxes`), points (useful for backbone angles; `w points`), lines (useful for distances; `w lines`) and smoothed lines (useful for almost everything else; `w lines s bezier`).



An example graph created using Gnuplot.

The following was the script used to create the above image.

```
set encoding iso_8859_1
#set term postscript enhanced color font "Arial,24";
set terminal pngcairo enhanced color font "Arial,24" size 100
0,750;

set xlabel "Value ({\260})"
set ylabel "Value ({\305})"
set xrange [1:80]
set key outside right Left reverse width 2 height 1 font "Aria
l,18" maxrows 3

set xtics ("Start" 0, 20, "Half" 40, 60, "End" 80) border nomir
ror out;
set x2tics border nomirror out rotate by 15 ("Etwas" 55)

set arrow 1 from first 50,400 to first 60,400 lw 2 heads
set label "Hi Mom" at 50,600 font ",18"

set output "test-image.png";
plot "test.dat" u ($1):($2) w boxes t "Boxes" lw 4, \
"test.dat" u ($1):($2) w points t "Points" lw 4, \
"test.dat" u ($1):($2) w lines s bezier t "Lines" lw 4;
```

Something that's been danced around a little bit so far is the idea of using transparent colors with pngcairo. Because of how gnuplot plots information like someone makes a sandwich (bread, cover the bread with peanut butter, hide the peanut butter with bananas, hide the bananas with more bread), data sets can be completely obscured from the seen plot. That's where transparency comes in!

Colors can be made transparent by adding two characters to the beginning of the color's HEX code. These two numbers correspond to the percentage of transparency. The thought process is flipped for gnuplot, though. If you use what would be 100%, that means that it'll be 100% transparent, and not 100% opaque. The characters to add to the HEX code are based on multiplying the 255 rgb scale number by the percentage, and then converting that number to hexadecimal. Lucky for you, I've put them in the below table (page 162) for you.

Table: Hexadecimal transparency characters

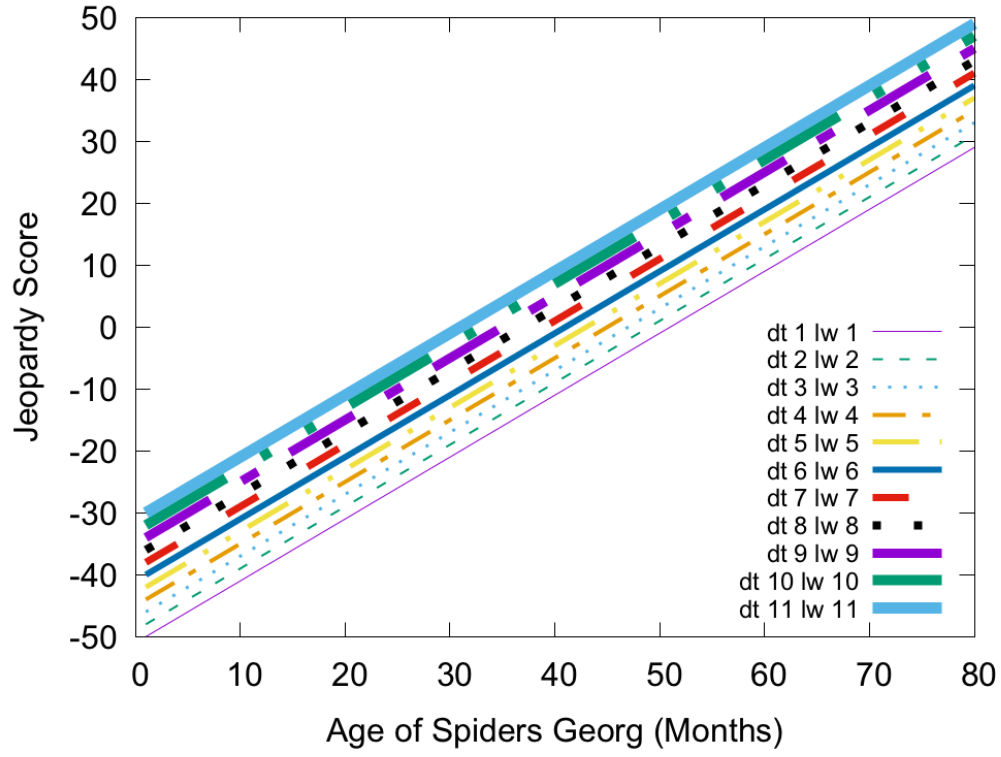
Where 100% is 100% transparent and 0% transparent has a "00" HEX code.

Percentage (%)	HEX	(%)	HEX	(%)	HEX	(%)	HEX
100	FF	75	BF	50	80	25	40
95	F2	70	B3	45	73	20	33
90	E6	65	A6	40	66	15	26
85	D9	60	99	35	59	10	1A
80	CC	55	8C	30	4D	5	0D

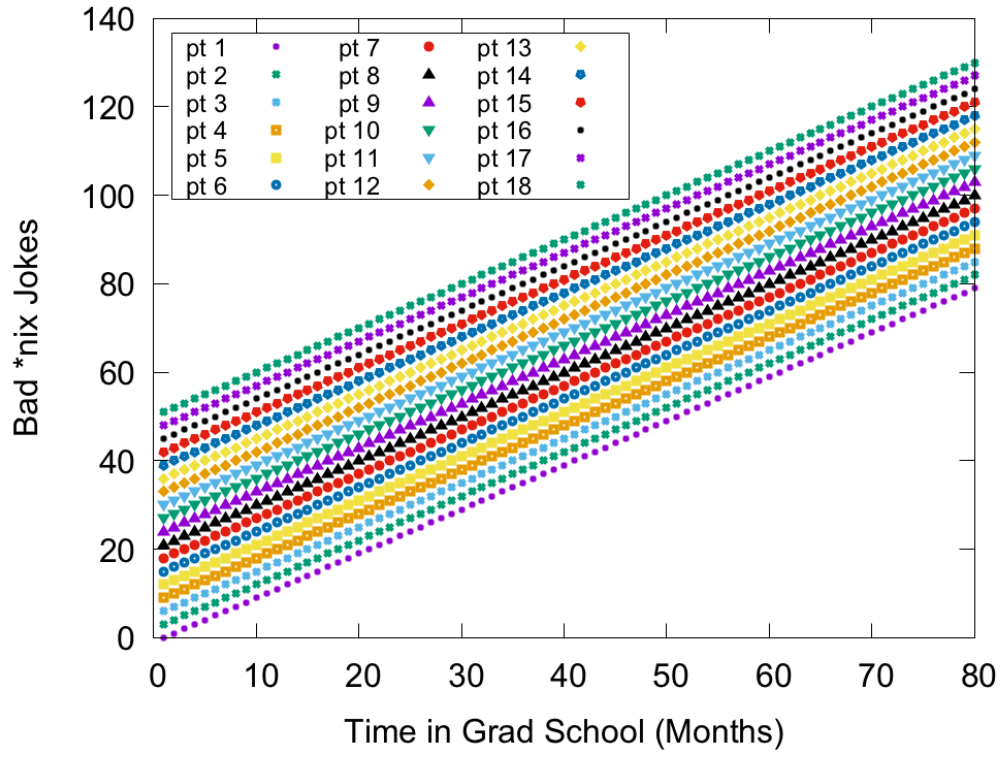
One working set of commands for incorporating transparency can be seen below. The box is set to allow transparency, then circles are used. The first thing plotted is completely opaque, so it's HEX code starts with `00`. The next thing plotted has a specified solid fill of 15%, but it also has 20% transparency given by a HEX value of `33`.

```
set style fill transparent solid 1.0 noborder
set style circle radius 0.02
plot ... lt rgb "#0000FF7F", \
... lt rgb "#339400D3" fs solid 0.15, \
```

Different weights and dash or point types can be used through setting `lw` (line weight) and either `dt` (dashtype) or `pt` (point). Again, gnuplot tends to cycle through a cycle of predetermined numbers and colors, but you can respecify anything in the plot command. You can even define dashtypes with any string containing dots, hyphens, underscores, and spaces (`dt` " `.. -- __` "). There are also plenty of point types (something like 75). Some examples of the different weights, dashes, and points are shown in the following images.



Different line weights and dash types.



Different point types.

An abbreviated example of the script to generate both graphs would be:

```
set key bottom right font "Arial,18"

set output "test-line.png";
plot "test-line.dat" u ($1):($2-10) w lines s bezier t "dt 1 l
w 1" lt 1 lw 1 dt 1, \
"test-line.dat" u ($1):($2-8) w lines s bezier t "dt 2 lw 2" l
t 2 lw 2 dt 2;

set key box top left font "Arial,18" width -1 maxrows 6

set output "test-point.png";
plot "test-line.dat" u ($1):($2+40) w points t "pt 1" pt 1 lw
4, \
"test-line.dat" u ($1):($2+43) w points t "pt 2" pt 2 lw 4;
```