

Appendix to the Dissertation:

Various UNIX and Perl scripts used in this research

A brief description of each script:

make14mer.com – generates a 14-mer from a 7-mer input PDB file

racc.pl – rare codon calculator, to predict if rare Arg, Leu, Ile, or Pro codons may be a problem in protein over-expression in *E. coli*

make_homology_model.pl – to create molecular replacement homology models from an input sequence alignment and PDB file

scripted_glr.sh, alter.pl, and process_bigrun.pl –
to calculate cross-rotation functions that are systematically varied over the integration radius and resolution limits

map_conservation_to_B.pl – to map sequence conservation values to the B-factor fields of a PDB file

symm.csh and write_pdbset.pl –
calculates all non-crystallographic symmetry transformations and expands the monomeric subunit into the multimer (matrix of all pairwise RMSDs between subunits of the multimer is output as well)

“/asimov2/users/cam/REDID_1108/14MER_MODEL/make14mer.com”

```
#!/bin/csh -f

# This c-shell script builds a 14-mer model given the most recent 7-mer structure
# available
#
# The input consists of:
#
# (1) A PDB file containing most recent 7-mer structure ("hept_exp_from_mono6.pdb")
# (2) A PDB file containing the atoms from which the center of gravity and rotational
#     axes will be calculated ("cys7sg.pdb")
# (3) Command line-input values for the desired translation distance (in Angstroms)
#     and rotation angle (in degrees)
# (4) NOTE: The input PDB files must contain the "CRYST" "SCALE" and "ORIGX" lines
#         The chains must be labelled by distinct chainid's, starting from 'A'.
#
# The output consists of:
#
# (1) A PDB file containing the input 7-mer translated so that its desired COG lies at
#     the origin ("translatel.pdb")
# (2) A PDB file of the rotated heptamer ("lsqkab.pdb")
# (3) A PDB file of the composite 14-mer ("make14mer_output.pdb")
# (4) plus some log files if the last clean-up step is commented out

##### INITIALIZE VARIABLES TO NULL #####

set v1; set v2; set v3
set norm_v1; set norm_v2; set norm_v3
set count
set dist
set angle

##### ASK FOR THE FINAL TRANSLATION DISTANCE (IN ANGSTROMS) IF NOT INPUT ON COMMAND LINE
#####
##### ASK FOR THE FINAL DESIRED ROTATION ANGLE (IN DEGREES) IF NOT INPUT ON COMMAND LINE
#####

if ( ${1} == "" ) then
    echo ""; echo "Program needs command line-input TRANSLATION DISTANCE \\!\\!"; echo ""
    kill -9 $$
else set dist = $argv[1]; echo ""; echo "translation distance will be: $dist angstroms"
endif

if ( ${2} == "" ) then
    echo ""; echo "Program needs a command line-input ROTATION ANGLE \\!\\!"; echo ""
    kill -9 $$
else set angle = $argv[2]; echo ""; echo "rotation angle will be: $angle degrees"
endif

##### CALCULATE THE CENTER OF GRAVITY FOR THE SG atoms OF THE CYS RING OF THE 7-MER #####

pdbset xyzin cys7sg.pdb << eof-1 > cog.log
SPACEGROUP C2
COM
eof-1

##### TRANSLATE INPUT 7-MER SO THAT CENTER OF GRAVITY OF CYS SG LIES AT ORIGIN #####

echo "#!/bin/csh -f" > translatel.com
echo "pdbset XYZIN almost_pruned.pdb XYZOUT translatel.pdb << eof-2 > translatel.log" >>
translatel.com
echo "" >> translatel.com
echo "TRANSFORM 1 0 0 -" >> translatel.com
echo "      0 1 0 -" >> translatel.com
echo "      0 0 1 -" >> translatel.com
```

```

echo -n " " >> translatel.com
echo -n "-"(grep " Center of Mass:" cog.log) | awk '{print $4}' ` >> translatel.com
echo -n " " >> translatel.com
echo -n "-"(grep " Center of Mass:" cog.log) | awk '{print $5}' ` >> translatel.com
echo -n " " >> translatel.com
echo -n "-"(grep " Center of Mass:" cog.log) | awk '{print $6}' ` >> translatel.com
echo -n " " >> translatel.com
echo "" >> translatel.com
echo "SPACEGROUP C2" >> translatel.com
echo "eof-2" >> translatel.com

chmod 755 translatel.com
translatel.com

##### USE AL_MOLEMAN TO SPLIT THE 7-MER INTO MONOMERS FOR UPCOMING USE IN ALIGN_V2 #####

al_moleman << eof > al_moleman.log

translatel.pdb
split
split_
quit
eof

##### CALCULATE AND NORMALIZE THE VECTOR ABOUT WHICH ROTATION WILL BE PERFORMED #####

set v1 = `grep SG translatel.pdb | head -1 | awk '{print $7}' `
set v2 = `grep SG translatel.pdb | head -1 | awk '{print $8}' `
set v3 = `grep SG translatel.pdb | head -1 | awk '{print $9}' `

echo ""
echo "unnormalized rotation axis = "$v1 $v2 $v3

set count=0

foreach z ($v1 $v2 $v3)
set count=`expr $count + 1`
echo "scale=5" > bc.script
echo "l=sqrt(`echo $v1`^2 + `echo $v2`^2 + `echo $v3`^2)" >> bc.script
echo "i=$z/l" >> bc.script
echo "i" >> bc.script
echo "quit" >> bc.script

if ($count == 1) then
set norm_v1 = `bc bc.script`
endif
if ($count == 2) then
set norm_v2 = `bc bc.script`
endif
if ($count == 3) then
set norm_v3 = `bc bc.script`
endif

end

echo ""
echo "normalized rotation axis = "$norm_v1 $norm_v2 $norm_v3; echo " ";

##### CALCULATE THE VECTOR ALONG WHICH THE TRANSLATION WILL TAKE PLACE
##### (i.e. THE 7-FOLD ROTATION AXIS)
##### USING ALIGN_V2 PROGRAM. NOTE THAT IT'S ALREADY NORMALIZED IN THE ALIGN_V2 OUTPUT
#####

align_v2 << eof > split.log

1

```

```

split_a.pdb

PDB
split_b.pdb

PDB
AUTO
split_a_to_b.rot
split_a_to_b.show
stop
eof

set norm_t1 = `grep direction split.log | awk '{print $4}' `
set norm_t2 = `grep direction split.log | awk '{print $5}' `
set norm_t3 = `grep direction split.log | awk '{print $6}' `

echo
echo -n "normalized vector along which translation will be performed (i.e. 7-fold axis) = "
"
echo $norm_t1 $norm_t2 $norm_t3

##### NOW SCALE THE TO-BE-APPLIED TRANSLATION VECTOR BY THE COMMAND LINE-INPUT AMOUNT
#####

bc << eof > temp1
-`echo $dist` * $norm_t1
quit
eof
set use_t1 = `awk '{print $1}' temp1`

bc << eof > temp2
-`echo $dist` * $norm_t2
quit
eof
set use_t2 = `awk '{print $1}' temp2`

bc << eof > temp3
-`echo $dist` * $norm_t3
quit
eof
set use_t3 = `awk '{print $1}' temp3`

echo ""
echo -n "the scaled translation vector that will be applied = "
echo $use_t1 $use_t2 $use_t3; echo " "; echo " ";

##### PERFORM DESIRED ROTATION & TRANSLATION ALONG CALCULATED AXES VIA DIRECTION COSINES
#####

echo "#!/bin/csh -f" > lsqkab.com
echo "lsqkab XYZIN2 translate1.pdb XYZOUT lsqkab.pdb << eof-z > lsqkab.log" >> lsqkab.com
echo "ROTATE DCS \"$norm_v1\" \"$norm_v2\" \"$norm_v3\" \"${angle}\" >> lsqkab.com
echo "TRANSLATE \"$use_t1\" \"$use_t2\" \"$use_t3\" >> lsqkab.com
echo "END" >> lsqkab.com
echo "eof-z" >> lsqkab.com

chmod 755 lsqkab.com
lsqkab.com

##### END OF SCRIPTS, so...#####
##### clean up stuff #####

cat translate1.pdb lsqkab.pdb > make14mer_output.pdb

rm split* temp* cog.log XYZOUT crap* al_moleman.log
rm translate1.com translate1.log bc.script lsqkab.com lsqkab.log

```

“/asimov2/users/cam/RAC/racc.pl”

```
#!/joule2/programs/bin/perl

# Calculates the number of rare (for E. coli) Arg, Leu, Ile, and Pro codons in input
# nucleic acid sequence file, and number of consecutive rare Arg, Leu, Ile, and Pro
# codons if > 1

# Ask for the nucleic acid sequence file name (if it was not entered on the command line)
# and open it:

if ($ARGV[0] eq undef)
{
    print "Enter the name of the nucleic acid sequence file (in GCG format):";
    chomp ($naseq = <STDIN>);
    if ($naseq =~ /\./)
        { $output_file = $naseq . "..racc";
          $output_file =~ s/\.*?\.//g; }
    else { $output_file = $naseq . ".racc"; }
    print "\nA file named \"$output_file\" will contain the results that follow\n\n";
    open (OUTFILE, ">$output_file") || die "Can't open \"$output_file\" for writing\n\n";
}

else { chomp ($naseq = $ARGV[0]); }

open (NASEQ, $naseq) || die "Can't open sequence file: $naseq\n";

# check that the NA sequence file is in GCG format:

chomp ($okornot = <NASEQ>);
unless ($okornot =~ /!!NA_SEQUENCE\b/)
{
    print "\nThe nucleic acid sequence file \"$naseq\" is not in GCG format!\n";
    if ($ARGV[0] eq undef)
        { print OUTFILE "\nNucleic acid sequence file \"$naseq\" is not in GCG format!\n"; }
}

# read in and process the NA sequence:

@seq = <NASEQ>;

for ($i = 0; $i <= $#seq; $i++)
{
    if (($seq[$i] =~ /[bdefhijklmnopqrstuvwxyz]/i) |
        !($seq[$i] =~ /\d/) | !($seq[$i] =~ /[a-z]/i))
        { $seq[$i] = "";}
}

$whole_seq = join("", @seq);
$whole_seq =~ s/[^a-z]//ig;
#$whole_seq =~ s/(\d)|(\s)//g;  an almost equivalent way to do it

# calculate properties of interest:

$seq_length = rindex($whole_seq, /\w/);
print "\nFor the following sequence from the input file \"$naseq\":\n";
print "\n$whole_seq\n";
print "\nThe length is: $seq_length nucleotides\n\n";

if ($ARGV[0] eq undef)
{
    print OUTFILE "For following sequence from input file \"$naseq\":\n";
    print OUTFILE "\n$whole_seq\n";
    print OUTFILE "\nThe length is: $seq_length nucleotides\n";
}
}
```

```

#warn user if DNA sequence isn't an ORF (i.e. multiple of 3 in length)
unless ($seq_length % 3 == 0)
{
  warn "\nNote: the sequence file \"$naseq\" is not an ORF\n";
  if ($ARGV[0] eq undef)
    {print OUTFILE "\nNote: the sequence file \"$naseq\" is not an ORF\n";}
}

# initialize rac hash variables:

%rac = ();

$rac{rac_1x} = 0;           # number of single rare Arg codons
$rac{rac_2x} = 0;           # number of tandem rare Arg codon repeats
$rac{rac_3x} = 0;           # number of triple rare Arg codon repeats

@rac{10, 11} = ("(aga)|(agg)|(cga)", 0); # no. single rare Arg codons
@rac{10, 12} = ("(agg)|(aga)|(cga)", 0); # no. tandem rare Arg codon rpt
@rac{10, 13} = ("(agg)|(aga)|(cga)", 0); # no. triple rare Arg codon rpt

@rac{20, 21} = ("cta", 0);           # number of single rare Leu codons
@rac{20, 22} = ("cta", 0);           # number of tandem rare Leu codon repeats
@rac{20, 23} = ("cta", 0);           # number of triple rare Leu codon repeats

@rac{30, 31} = ("ata", 0);           # number of single rare Ile codons
@rac{30, 32} = ("ata", 0);           # number of tandem rare Ile codon repeats
@rac{30, 33} = ("ata", 0);           # number of triple rare Ile codon repeats

@rac{40, 41} = ("ccc", 0);           # number of single rare Pro codons
@rac{40, 42} = ("ccc", 0);           # number of tandem rare Pro codon repeats
@rac{40, 43} = ("ccc", 0);           # number of triple rare Pro codon repeats

# parse the sequence from $whole_seq into codons and count away:

for ($i = 1; $i <= ($seq_length/3); $i++)
{
  $codon[$i-1] = substr($whole_seq, ($i-1)*3, 3);

  if ($codon[$i-1] =~ /(agg)|(aga)|(cga)/i)
  { $rac{rac_1x}++; push(@rac_1x_positions, $i); }

  if ($i >= 2)
  {
    if (($codon[$i-1] =~ /(agg)|(aga)|(cga)/i) & ($codon[$i-2] =~ /(agg)|(aga)|(cga)/i))
    { $rac{rac_2x}++; push(@rac_2x_positions, $i); }
  }

  if ($i >= 3)
  {if (($codon[$i-1] =~ /(agg)|(aga)|(cga)/i) & ($codon[$i-2] =~ /(agg)|(aga)|(cga)/i)
      & ($codon[$i-3] =~ /(agg)|(aga)|(cga)/i))
    { $rac{rac_3x}++; push(@rac_3x_positions, $i); }
  }
}

##### now repeat the same for Leu, Ile, and Pro codons: #####

for ($a = 10; $a <= 40; $a += 10)
{

for ($i = 1; $i <= ($seq_length/3); $i++)

{ $a1 = $a + 1; $a2 = $a + 2; $a3 = $a + 3;

  if ($codon[$i-1] =~ /($rac{$a})/i)

```

```

{ $rac{$a1}++; push(@posit_1x, $i);
}

if ($i >= 2)
{
if (($codon[$i-1] =~ /($rac{$a})/i) & ($codon[$i-2] =~ /($rac{$a})/i))
{ $rac{$a2}++; push(@posit_2x, $i);
}
}

if ($i >= 3)
{if (($codon[$i-1] =~ /($rac{$a})/i) & ($codon[$i-2] =~ /($rac{$a})/i)
& ($codon[$i-3] =~ /($rac{$a})/i))
{ $rac{$a3}++; push(@posit_3x, $i);
}
}
}

push(@posit_1x, ("$rac{$a}" . $a1/10));
push(@posit_2x, ("$rac{$a}" . $a2/10));
push(@posit_3x, ("$rac{$a}" . $a3/10));

}

for ($i = 1; $i <= ($seq_length/3); $i++)
{$codon[$i-1] = $codon[$i-1] . " "};

for ($i = 0; $i <= ($rac{rac_1x}-2); $i++)
{$rac_1x_positions[$i] = $rac_1x_positions[$i] . ", "};
for ($i = 0; $i <= ($rac{rac_2x}-2); $i++)
{$rac_2x_positions[$i] = $rac_2x_positions[$i] . ", "};
for ($i = 0; $i <= ($rac{rac_3x}-2); $i++)
{$rac_3x_positions[$i] = $rac_3x_positions[$i] . ", "};

print "Number of total single rare Arg codons: $rac{rac_1x}\n";
if ($rac{rac_1x} > 0)
{print "occurring at codons: @rac_1x_positions \n";}

print "Number of tandem rare Arg codon double repeats: $rac{rac_2x}\n";
if ($rac{rac_2x} > 0)
{print "occurring at codons: @rac_2x_positions \n";}

print "Number of tandem rare Arg codon triple repeats: $rac{rac_3x}\n\n";
if ($rac{rac_3x} > 0)
{print "occurring at codons: @rac_3x_positions \n\n";}

##### preliminary output formatting of rare codons for Leu, Ile, and Pro: #####

print "Too lazy to beautify this new part right now...Results are in order for
Arginine, Leucine, Isoleucine, and Proline, respectively (delimited by numbers
1.1, 2.1, 3.1, 4.1 for singles; 1.2, 2.2, 3.2, 4.2 for doubles; etc.)\n";

print "\nSingle rare codons at positions:\n@posit_1x \n";
print "\nDouble rare codons at positions:\n@posit_2x \n";
print "\nTriple rare codons at positions:\n@posit_3x \n";

##### END OF RaCC SCRIPT #####

```

“/asimov2/users/cam/UTILITIES/make_homology_model.pl”

```
#!/joule2/programs/bin/perl

# This program begins with a sequence alignment, of the sort produced by the GCG
# "bestfit" program. It tabulates the positions that are strongly conserved.
# Then, all strongly conserved residues (i.e. paired by either ':' or '|' in the
# GCG output), ALAs, and GLYs are left alone, while all other residues are
# truncated to ALA as in the "make_poly_ala.pl" program. Extra residues in
# the homology model probe protein that correspond to gaps in the actual protein
# of interest will be deleted.
#
# NOTE: It is assumed that the input PDB file has been edited to contain only 1
# copy of each unique chain that exists in the sequence alignment; and that the
# beginning and ending residues in the PDB file correspond to those at the beginning
# and end of the sequence aligned homology model protein (even though all other
# residues in the PDB file will be unaltered).

# input all of the file names, check that they're accessible

print "\nEnter the name of the GCG .pair sequence alignment file: ";
chomp ($sa_input = <STDIN>);

open (SA_INFILE, $sa_input) || die "Can't open sequence alignment file
\"$sa_input\"\n\n";

print "\nEnter 't' if the homologous protein of known structure is the top
sequence or 'b' if it's the bottom sequence of the sequence alignment: ";
chomp ($top_or_bot = <STDIN>);

print "\nEnter the name of the PDB input file: ";
chomp ($pdb_input = <STDIN>);
if ($pdb_input =~ /\./) {$pdb_output = $pdb_input; $pdb_output =~
s/\./_HOMOLOGY_MODEL./;}
else {$pdb_output = $pdb_input . "_HOMOLOGY_MODEL.pdb";}

# process the .pair file to extract conserved residues and those that should be
# deleted

@strings = <SA_INFILE>;

for ($i = 0; $i <= $#strings; $i++)
{
    if ( $strings[$i] =~ /\s+\d+\s+[A-Ya-y\.\.]+\s+\d+\s+/ )    { $index = $i-1;
        last; }
}

for ($j = 0; $j < $index-1; $j++) { shift(@strings); }

$k = $#strings;

if ($k < 5) {$num_row_sets = 1;}
else {$num_row_sets = $k / 4;}

shift(@strings);

#for ($x = 1; $x <= $k; $x++) {print "\$strings[$x] = $strings[$x]";}
#print "\n\n";

if ($num_row_sets == 1) { for ($n=1; $n<=3; $n++) {$prot[$n] = $strings[$n];} }
else
{
```

```

    for ($n = 1; $n <= 3; $n++)
    {
        $prot[$n] = $strings[$n];

        if ($n == 2)
        {
            for ($q = 1; $q < $num_row_sets; $q++)
            { $z = $n + (4 * $q); substr($strings[$z], 0, 9) = "";
chop($prot[2]);
                $prot[2] = $prot[2] . $strings[$z]; }
            }

            else {
                for ($q = 1; $q < $num_row_sets; $q++)
                { $z = $n + (4 * $q);
                    $prot[$n] = $prot[$n] . $strings[$z]; }
            }
        }
    }

for ($y=1; $y<=3; $y++) { $prot[$y] =~ s/(\n|\r)//g; chomp($prot[$y]); }

$prot[1] =~ s/ \d+\s\d+ //g;
$prot[3] =~ s/ \d+\s\d+ //g;

if ($top_or_bot eq "b") {$homo_prot = $prot[3]; $your_prot = $prot[1]; }
else {$homo_prot = $prot[1]; $your_prot = $prot[3]; }
$alignment = $prot[2];

# extract beginning and ending residue numbers for new MR search model PDB file:
if ($homo_prot =~ /\s+(\d+)\s+.* (\d+)\s*/) { $res_begin = $1; $res_end = $2; }

else {die "\nSomething is really wrong-- there are no residue numbers in homo_prot\n\n";}

# print out current status:

$homo_prot =~ s/\d/ /g;
$your_prot =~ s/\d/ /g;

substr($homo_prot, 0, 9) = "";
substr($your_prot, 0, 9) = "";
substr($alignment, 0, 9) = "";

# Turn the following lines on to print out the input, re-formatted for the next step:
#print "your_prot =$your_prot\n";
#print "alignment =$alignment\n";
#print "homo_prot =$homo_prot\n";
#print "\n\nBegin residue = $res_begin\n";
#print "\n\nEnd residue = $res_end\n\n";

@your_prot = split(/, $your_prot);
@alignment = split(/, $alignment);
@homo_prot = split(/, $homo_prot);

# Now make the comparisons and write out the verdict array:

for ($w = 0; $w <= $#homo_prot; $w++)
{
    if ($homo_prot[$w] eq ".") { $verdict[$w] = "correct_the_index"; next;}

    if (($homo_prot[$w] ne ".") & ($your_prot[$w] eq ".")) {$verdict[$w] = "delete";next;}

    if (($alignment[$w] eq ":" ) | ($alignment[$w] eq "|")) {$verdict[$w] = "conserve";next;}

    if (($homo_prot[$w] =~ /g/i) | ($homo_prot[$w] =~ /a/i)) {$verdict[$w]="conserve";next;}

    $verdict[$w] = "truncate";
}

```

```

}

for ($j=0; $j <= $#homo_prot; $j++) { if ($homo_prot[$j] =~ /[a-zA-Y\.\_]/) { $index_end =
$j; } }

#for ($t=0; $t<=$#homo_prot; $t++)
#{print"\$homo_prot[$t]=$homo_prot[$t] \t $verdict[$t] \n";}
#print"\nEnd value = $index_end\n\n";

# this next section corrects the index values so that the final_homo and final_verdict
array indices
# correspond to the actual homology model protein residue numbers, starting from 1 at the
N'

for ($i = 0; $i <= $index_end; $i++)
{
  if ($verdict[$i] eq "correct_the_index") {$index_corrector++; next;}

  $correct_index = $i + $res_begin - $index_corrector;

  $final_homo_prot[$correct_index] = $homo_prot[$i];
  $final_verdict[$correct_index] = $verdict[$i];
}

# turn the next section on to print out the verdict array, i.e. a list of the verdicts
# (delete, truncate, or conserve) for each homology model protein residue

#for ($t=0; $t<=$#final_homo_prot; $t++)
#{print"\$final_homo_prot[$t]=$final_homo_prot[$t] \t $final_verdict[$t] \n";}

# process the PDB file:

open (PDB_INFILE, $pdb_input) || die "Can't open PDB file \"$pdb_input\"\n\n";
open (PDB_OUTFILE, ">$pdb_output") || die "Can't open PDB file \"$pdb_output\" for
writing\n\n";

print "\n\nOutput file \"$pdb_output\" contains the homology model coordinates.\n\n";

while (<PDB_INFILE>)
{
  if (!/^ATOM +\d+/) {print PDB_OUTFILE "$_"; }

  else

    {

      $_ =~ /\s+(\d+)\s+.*\s+(\d+)\s+/ ; $res_num = $2;

      if ($final_verdict[$res_num] eq "delete") {next;}
#the delete verdict

      if ($final_verdict[$res_num] eq "conserve") {print PDB_OUTFILE "$_"; next;}
#the conserve verdict

      if ( ($final_verdict[$res_num] eq "truncate") & ($_ =~ /^ATOM[ 0-9]{9}(N |CA|CB|C
|O )/) )
        {s/(VAL|LEU|ILE|MET|PRO|PHE|TRP|SER|THR|ASN|GLN|TYR|CYS|LYS|ARG|HIS|ASP|GLU)/ALA/;
        print PDB_OUTFILE "$_"; next;}

    }
}

```

“/asimov2/users/cam/PAsurE_xtal/GLRF/scripted_glr.sh”

```
#!/bin/csh -f

# a script to automatically perform several GLRF cross rotation function runs at several
# values of 3 user-input parameters (integration radii, resolution windows, and models)
# IMPORTANT: this works only in conjunction with the program GLRF and the perl script
#             "alter.pl", and assumes you've already synthesized structure factors for the
#             various models (e.g., with sfcalf.com) in a file called "models.cal"
#
# command line syntax is:
# " >scripted_glr.sh model_file rad_low rad_high width_of_resol_win "
#
# The output is a file containing the top 5 peaks from each RF run...

set rad_low rad_high resol_win models_file
set curr_rad curr_res_low curr_res_high curr_model

# set the command line-input variables

set models_file = $argv[1]
set rad_low = $argv[2]
set rad_high = $argv[3]
set resol_win = $argv[4]

# regurgitate the user's command line input. don't worry about checking for
# null inputs and setting default values.

echo ""
echo "VARIATION OF THE 1st PARAMETER (integration radius):"
echo "*****"
echo "GLRF will use a minimum radius of:"    $rad_low
echo "GLRF will use a maximum radius of:"  $rad_high
echo "GLRF will use these radii incremented by 1.0 Angstroms"
echo ""
echo ""
echo "VARIATION OF THE 2nd PARAMETER (resolution):"
echo "*****"
echo "GLRF will calculate RFs over a sliding window of resolutions,"
echo "starting at a high resolution limit of 3.5 Angstroms, incrementing"
echo "by 0.5 Angstroms, and with a fixed window width of:"  $resol_win "Angstroms"
echo ""
echo ""
echo "VARIATION OF THE 3rd PARAMETER (MR model):"
echo "*****"
echo "The filename containing the model names is:"  $models_file
echo "and the models specified in this file are:"
echo ""
cat $models_file
echo ""
echo ""

# now the 3 nested loops that are the core of this script:

set allmodels = `cat $models_file`

bc << eof_bc > temp999
`echo $resol_win` * 10
quit
eof_bc

set resol_win = `awk '{print $1}' temp999 `
set absolut_hi_res = ` expr 35 + $resol_win `

foreach curr_model ( ${allmodels} )
```

```

set prefix = `(echo $curr_model | awk '{split($1,a,"."); print a[1]}')`
set curr_rad = $rad_low

while ($rad_low <= $curr_rad && $curr_rad <= $rad_high)

    set curr_res_low = 35; set curr_res_high = `expr $curr_res_low + $resol_win`

    while (35 <= $curr_res_low && $curr_res_low <= $curr_res_high)

        echo "running alter.pl ${curr_model} radius=${curr_rad}
res.range=${curr_res_low}-${curr_res_high} ..." >> big_run.log
        rm scripted_cross_rf.prt
        rm cross_rf_temp.com
        alter.pl ${curr_model} radius=${curr_rad} res.range=${curr_res_low}-
${curr_res_high}
        chmod 755 cross_rf_temp.com
        cross_rf_temp.com
        egrep "^          +[0-9]+ +[0-9]+ +[0-9]+ +[0-9]+ +[0-9]+\"
scripted_cross_rf.prt | awk '{print $12}' | head -5 >> big_run.log
        set curr_res_low = `expr $curr_res_low + 5`
        set curr_res_high = `expr $curr_res_high + 5`
        if ($curr_res_low >= $absolut_hi_res) break

    end

    set curr_rad = `expr $curr_rad + 1`
    echo ""

end

echo "" ; echo "*****" ; echo ""

end

rm temp999

```

“/asimov2/users/cam/PAsurE_xtal/GLRF/alter.pl”

```
#!/joule2/programs/bin/perl

open (IN, "scripted_cross_rf.com") || die "Cannot open file \"scripted_cross_rf.com\"\n";
open (OUT, ">>cross_rf_temp.com");

$model = "$ARGV[0]";
$radius = "$ARGV[1]"; $radius =~ /(\d+)/; $radius = $1;
$res_range = "$ARGV[2]"; $res_range =~ /(\d+)-(\d+)/; $res_hi = $1/10; $res_low = $2/10;

print "$model\n$radius\n$res_hi\n$res_low\n" ;

while (<IN>)
{
  chomp ($_);
  if ($_ =~ /aobsfile/)
  {
    $new = "aobsfile MODELS\"/$model";
    print OUT "$new\n";
    next;
  }

  if ($_ =~ /resolution/)
  {
    $new = "resolution $res_low $res_hi";
    print OUT "$new\n";
    next;
  }

  if ($_ =~ /radius/)
  {
    $new = "radius $radius";
    print OUT "$new\n";
    next;
  }

  print OUT "$_\n";
  next;
}
```

“/asimov2/users/cam/PAsurE_xtal/GLRF/process_bigrun.pl”

```
#!/joule2/programs/bin/perl

# this script processes the output from scripted_glrif.sh into a format good for excel or
some
# other spreadsheet.

$infile = $ARGV[0];

open (IN, $infile) || die "Cannot open file \"$infile\"\n";

print "MODEL\t\tInteg_radius\tHI RESOL LIM\tTOP 5 PEAKS in order\tLARGEST diff\n" ;
print "*****\n";

while (<IN>)
{
  chomp ($_);
  if ( $_ =~ /running alter\.pl (\w+\\.cal) radius=(\d+) res\.range=(\d+)-\d+/ )
  {
    $model = $1; $radius = $2; $res_high_lim = $3/10;
    $verdict = 1; $peaks = ""; next ;
  }

  else {
    $peaks = $peaks ."\t". $_ ; $verdict++;
    if ($verdict == 6)
      { @top5 = split(/\s+/, $peaks);
# Now calculate the diff btwn peak 1 & peak 2,
# since this is what's really interesting for RF
      $diff = $top5[1] - $top5[2];
      print "$model\t\t$radius\t\t$res_high_lim\t\t$peaks\t\t$diff\n"; next ;
      }
  }

  next ;
}
}
```

“~cam/PAsurE_xtal/MAD_Jan2002/STRUC_ANAL/map_conservation_to_B.pl”

```
#!/joule2/programs/bin/perl
#
# This is a script that reads in 2 input files: argv[0] is the PDB file
# and argv[1] is the file tabulating conserved residues in the format:
#
# "res#,ss(super strong)|s(trong)|m(edium)|w(eak)" ...
#
# where "res#" is the residue # and "superstrong", "strong", "medium",
# or "weak" specify how strongly that site is conserved.
# The output is a PDB file with all of the B-factors flattened to 20.00,
# except for the "superstrong", "strong", "medium", or "weak" sites, which
# are assigned B-values of 90.00, 70.00, 40.00, or 33.00, respectively.
#
# This is useful for programs like GRASP, which can color a surface by the
# "B-factor" field of the PDB file.
#
# Cameron Mura (July 2001)
# NOTE: make sure occupancies are 1.00 for any atoms with B-factors you want changed

$pdb_in = $ARGV[0];
$site_file = $ARGV[1];
$bfac = "";
@cons = "";
$site_line = "";

open (PDB, $pdb_in) || die "Cannot open file \"$pdb_in\"\n";

while (<PDB>)
{
    $line = $_;
    chomp ($line);
    $resnum = ""; $bfac = "";
    if ($line =~ /^(ATOM|HET)\s+\d+\s+[\w\*]+\s+\w\w\w [A-Z]\s+(\d+)\s+(\d+\.\d+)\s+(.*)/)
    {
        $resnum = $3; $bfac = "20.00";
        $counter = 0;
        open (SITES, $site_file) || die "Cannot open file \"$site_file\"\n";
        while (<SITES>)
        {if ($counter == 0)
        {
            $site_line = $_; chomp ($site_line); @cons = "";
            @cons = split(/,/, $site_line);
            if ($cons[0] == $resnum and $cons[1] eq "ss" )
                {$bfac = "95.00"; $resnum++; last;}
            if ($cons[0] == $resnum and $cons[1] eq "s" )
                {$bfac = "65.00"; $resnum++; last;}
            if ($cons[0] == $resnum and $cons[1] eq "m" )
                {$bfac = "45.00"; $resnum++; last;}
            if ($cons[0] == $resnum and $cons[1] eq "w" )
                {$bfac = "30.00"; $resnum++; last;}

            next;
        }
        }
        close (SITES);

        if ($bfac eq "") { $bfac = "20.00";}
        $crap = $5 . $bfac;
        print "$1$3$4$crap$7\n";
    }
}
```

“~cam/SMAP3/DATA/MAKE_28mer/symm.csh”

```
#!/bin/tcsh -f
# A script to calculate and expand NCS for any type of multimer ...
#
# For example, for the SmAP3 28-mer, subunit B was built most extensively. Now, one can
# use this script to create a 28-mer with each subunit being a copy of 'B'. The script
# figures out the NCS operators and applies them (via "align_v2" and CCP4 "pdbset"
# programs) to get several transformed PDB files ...
#
# command line syntax: >symm.csh inputPDBfile probechain
#
# where "inputPDBfile" is the COMPLETE multimer (e.g. SmAP3 28-mer), and
#       "probechain" is the CHAINID for the monomeric subunit you want to
#                   have copied 28-times over...
# Cameron Mura, May 2002

# get input ready
#
echo "";
set infile = $argv[1]
set probechain = $argv[2]
set prefix = `(echo $infile | awk '{split($1,a,"."); print a[1]}')`
set align_out = $prefix.align.out
echo "INPUT FILE IS:  "$infile; echo ""
echo "CHAINid USED TO EXPAND 28-mer: "$probechain; echo ""
echo "PREFIX FOR SPLIT FILES WILL BE: "$prefix ; echo ""
echo "PAIRWISE ALIGNMENTS WILL BE STORED IN: "$align_out ; echo ""

# split input file into several files separated by chainID
#
al_moleman << EOF_moleman > moleman.log.temp

${infile}
split
${prefix}_
quit
EOF_moleman

# re-name files so chainIDs get renumbered to ASCII equivalents
#
set num_files_dec = `ls ${prefix}_[0-9].pdb | wc -l`
set num_files_char = `ls ${prefix}_[a-z].pdb | wc -l`
set num_files_total = `expr $num_files_dec + $num_files_char`
echo "Number of files_decimal:  "$num_files_dec; echo ""
echo "Number of files_character:  "$num_files_char; echo ""
echo "Number of files_total:  "$num_files_total; echo ""

# must rename the decimal ones before the character ones, otherwise they'll
# definitely get overwritten by the a, b, c, etc...

set x = 1
while ($x <= $num_files_dec)
set rename_value = `expr $x + $num_files_char`
mv ${prefix}_${x}.pdb ${prefix}_${rename_value}.pdb
set x = `expr $x + 1`
end

set index1 = 1
while ($index1 <= $num_files_char)
set temp_ind = `expr $index1 + 96`
set to_rename2 = `perl -e 'print(chr($ARGV[0]),"\n")' $temp_ind`
mv ${prefix}_${to_rename2}.pdb ${prefix}_${index1}.pdb
set index1 = `expr $index1 + 1`
end
```

```

### HERE IS THE ALIGNMENT PART: #####
###
### include the following line to do both all_atom- and main_chain-mode alignments:
foreach mode (all_atom main_chain)
foreach mode (main_chain)

if ($mode == "all_atom") set option = 1
if ($mode == "main_chain") set option = 2

echo "====="; echo;
echo "ALIGNMENT MODE is:" \" $mode\" \"(align_v2 option $option)\ "; echo;
echo "====="; echo;

set i1 = 1

while ($i1 <= $num_files_total)

## comment-in the next line to do just an (upper) diagonal matrix of pairs:
#set i2 = $i1
# leave in THIS NEXT LINE to do full symmetric matrix worth of entries:
set i2 = 1

while ($i2 <= $num_files_total)

echo "for aligning" $i2 "to" $i1 "the rmsd is:"

### here's the "align_v2" part, don't mess with it:

align_v2 << eof > temp_${i1}_to_${i2}.log

${option}

${prefix}_${i1}.pdb

PDB
${prefix}_${i2}.pdb

PDB
AUTO
${prefix}_${i2}_to_${i1}.rot.temp
${prefix}_${i2}_to_${i1}.show.temp
stop
eof

cat ${prefix}_${i2}_to_${i1}.show.temp | grep 'rms deviation'
cat ${prefix}_${i2}_to_${i1}.show.temp | grep 'corresponds to a'
echo "-----"; echo;

##### apply the transformation matrix from CCP4:

write_pdbset.pl ${prefix}_${i2}_to_${i1}.show.temp ${prefix}_${i2}.pdb
${prefix}_${i2}_on_${i1}.new.pdb > pdbset_${i2}_to_${i1}.com
chmod 755 pdbset_${i2}_to_${i1}.com
pdbset_${i2}_to_${i1}.com > pdbset_${i2}_to_${i1}.log

echo "-----"; echo;
echo "Transforming subunit $i2 into position of $i1 (via PDBSET) ..."; echo;
echo "-----"; echo;

```

```

##### clean-up stuff #####
rm temp_${i1}_to_${i2}.log
rm pdbset_${i2}_to_${i1}.log
rm ${prefix}_${i2}_to_${i1}.show.temp
rm ${prefix}_${i2}_to_${i1}.rot.temp
rm pdbset_${i2}_to_${i1}.com
#####

set i2 = `expr $i2 + 1`

end

#####
# include following section to cat together the rotated files and originals into one
# final file:
#cat *to`echo $i1`_fr.rot gpa_mono`echo $i1`_fr.pdb > all_7rot_to`echo $i1`.pdb
#echo "*****";echo;
#echo Writing out PDB file of superimposed structures: \"all_7rot_to`echo $i1`.pdb\";
#echo
#echo "*****";echo;
#####

set i1 = `expr $i1 + 1`

end

#end

```

“~/cam/SMAP3/DATA/MAKE_28mer/write_pdbset.pl”

```
#!/joule2/programs/bin/perl

# A Perl script to convert align_v2-formatted .show file to pdbset.com file
# (just change CELL and SPACEGROUP cards below to generalize this) ...
# Is meant to work in conjunction with "symm.csh" script to expand one subunit
# of the SmAP3 28-mer into 28 copies...was written in generalizable format.
#
# Cameron Mura (May 2002)

$show_in = $ARGV[0];
$pdbin = $ARGV[1];
$pdbout = $ARGV[2];
$son = 0;

open (IN, "$show_in") || die "Cannot open file \"$show_in\"\n";
#open (OUT, ">temp_pdbset.com") || die "Cannot open file \"temp_pdbset.com\"\n";

while (<IN>)
{
    $line = $_;
    chomp ($line);
    if ($line =~ /Rotation matrix applied to latest set\: +([0-9\.\\-]+ +[0-9\.\\-]+
+([0-9\.\\-]+)\s*)/)
    {
        { $mat123 = $1; $son++; next; }
        if ($son == 1 & $line =~ / +([0-9\.\\-]+ +[0-9\.\\-]+ +[0-9\.\\-]+\s*)/)
            { $mat456 = $1; $son++; next; }
        if ($son == 2 & $line =~ / +([0-9\.\\-]+ +[0-9\.\\-]+ +[0-9\.\\-]+\s*)/)
            { $mat789 = $1; $son++; next; }
        if ($son == 3 & $line =~ / +([0-9\.\\-]+ +[0-9\.\\-]+ +[0-9\.\\-]+\s*)/)
            { $mat101112 = $1; $son++; next; }
        if ($son >= 4) { last; }
    }
}

# print out the PDBSET lines:
# (note that in the symm.csh script this is re-directed to a PDBSET .com file, which
# could be done at this stage instead (i.e. in this perl script), if necessary...)

print "#!\n/bin/csh -f\n";
print "pdbset xyzin $pdbin xyzout $pdbout <<eof-1\n";
print "transform\t$mat123 -\n";
print "\t\t$mat456 -\n";
print "\t\t$mat789 -\n";
print "\t\t$mat101112 \n\n";
print "CELL 83.322 172.428 148.108 90.000 89.986 90.000\n";
print "SPACEGROUP P21\n";
print "eof-1\n";

# THE END!
```