CHAPTER 8 "COMPUTER PROGRAMS"

This chapter is intended to be a resource guide for anyone interested in performing any of the calculations presented in chapters 1 through 6 of this thesis. These programs were written to give the biomolecular NMR spectroscopist the tools necessary to work with their NMR data and structural models.

The X-PLOR utilities are a series of scripts written to facilitate the use of the restrained molecular dynamics package, X-PLOR, with nucleic acids. These scripts use a common input file that allows one to easily generate and manipulate torsion angle, base pairing and planarity restraint files.

The hydrodynamic calculation scripts were developed for calculating both the rotational and translational diffusion rates of a hydrodynamical particle using spherical, elliptical and cylindrical models. This is useful for predicting the correlation time of a given biomolecule.

The inertia tensor program is used to predict the possible rotational anisotropy of a biomolecule by calculating the moment of inertia about the rotation reference frame. This is useful in deciding if a biomolecule is best described using one, two or three rotational correlation times.

Almost all programs in this chapter were written using the scripting language "PERL", and a quick discussion of why this language was used is in order. First, as an interpreted *scripting* language, PERL enjoys the enormous advantage of being almost completely portable between many computer types and operating systems. All the machine specific coding is done in the PERL interpreter, which is available for most major operating systems. The second advantage PERL has over other languages is that it

excels at text manipulations. PERL scripts can easily be a fraction of the size of their FORTRAN counterparts due to the many text manipulation tools built into the language.

The author wrote most of the computer programs presented in this chapter. The few exceptions (dm, noe_in and planar_make) are duly noted and are included only for the sake of completeness. Most of the programs were written specifically for the purposes needed by the author and may not be a general solution to for all situations. People are encouraged to use, and improve on, these programs.

8.1 X-PLOR utilities

X-PLOR is a software package that performs restrained molecular dynamics on biomolecules, and is the mechanism by which the distance and angular restraints determined by NMR are incorporated into a structural model. There are a number of restraint files that X-PLOR needs to successfully perform these calculations on nucleic acids.

Torsion angle restraints are used to adjust any of the seven torsion angles that define a nucleotide (cdih.dat). Distance restraints can be distilled into two types of files, the first is distances between heavy atoms to force hydrogen bonding between base pairs (hbond.dat) and the second is distances between protons (noe.dat). Finally, planarity restraints are used to force two nucleic acid bases to remain planar, typically between two base-paired nucleotides (planar.dat).

Necessarily, these input files are large, tedious and complex. A 20 nucleotide DNA, for instance would have 7x20 = 140 entries in the cdih.dat file that might look like that shown below (this entry defines the alpha torsion angle to have an angle of -46.8 degrees +/- 20). That would be nearly 5x140 = 700 lines of text.

```
assign (segid a and resid 1 and name O3')
      (segid a and resid 2 and name P  )
       (segid a and resid 2 and name O5')
      (segid a and resid 2 and name C5')
       1 -46.8  20  2
```

To encourage experimentation with the X-PLOR restraints, a number of programs have been written that perform the arduous task of building these input files. cdih_make build the torsion angle file, noe_hbond_make builds the hydrogen bonding file, dm/noe_in builds the noe.dat file and planar_make builds the planarity restraint file.

Additionally, cdih_measure has been written to examine the torsion angles of nucleic acid PDB files, useful for determining these angles after X-PLOR has created a structural model.

8.1.1 "seq" file format

Many of the scripts in this section utilize an input file called the "seq" (sequence) file. This file is intended to be a simple, yet powerful method of concisely defining the various parameters needed in order to generate XPLOR restraint files.

Two simple examples of this format are shown on the next page. In the first example, a standard B-form duplex DNA with sequence 5'-ATGC-3' is represented. Notice that the defaults (as defined by the Insight95 software package) for the B-form torsion angles and the default range of motion are defined at the beginning of this file.

In the next example, an A-form RNA monomer hairpin is being represented, 5'-UACAGUUUGUCUA-3'. Notice that the "ADDTONUM 20" line causes the numbering of the nucleotide to begin with the number 21, otherwise the numbering will begin with the number 1. Thus, the first uridine nucleotide will be named "U21". Some of the nucleotide letters are upper case, and some are not. If an upper-case nucleotide letter is found, the "default" torsion angles, as defined earlier in the file, are used for generating the "cdih.dat" X-PLOR restraint file. If a lower-case nucleotide letter is found, followed by the word 'none', then the X-PLOR restraint files will be built with no restraints for that nucleotide. Finally, if a lower-case letter is found followed by a list of the torsion angles as shown in the example for U26, that nucleotide will be given those specific torsion angle restraints (in this case, C2' endo sugar pucker, and more relaxed backbone angles).

B-form DNA "sequence" file for 5'-ATGC-3' duplex

```
! defaults (B-form):
alpha -46.8 20
beta -146 20
gamma 36.4 20
epsilon 155 20
zeta -95.2 20
chi -97.8 20
nu0 -4.2 10
nu1 24.9 10
nu2 -34.9 10
nu3 33.3 10
nu4 -18.3 10

segment a
A
T
G
C

segment b
G
C
A
T
```

A-form RNA "sequence" file for 5'-UACAGUUUCUGUA-3' hairpin

```
! defaults:
alpha -62 20
beta -180 20
gamma 47 20
epsilon -152 20
zeta -74 20
chi -166 20
nu0 3 10
nu1 -25 10
nu2 37 10
nu3 -36 10
nu4 21 10

ADDTONUM 20

U
A
C
A
g none

! U26: C2' endo sugar pucker.
u -62,45:-180,45:47,45:-152,45:-74,45:-166,45:-4, 10:25,10:-
34,10:33,10:-18,10

u none
u none
c none
U
G
U
A
```

8.1.2 cdih_make - create XPLOR dihedral files

This script creates XPLOR dihedral restraint files from the "sequence file" from section 7.2.1.

Syntax: **cdih_make < seq_file > cdih.dat**

In this example, the sample seq file for building the DNA 5'-ATGC-3' will be used and the first 30 lines of the XPLOR cdih.dat restaint file will be shown. Note, however, that the full length of this restaint file is ~400 lines. Also notice that the header information the cdih_make generates includes information on the length of the DNA, and the sequence. Because the input seq file contained the "segment a" and "segment b" lines, XPLOR segids are used in the atom definitions.

```
bass (lapham): [~/xplor/thesis]> cdih_make < atgc > cdih.dat
bass (lapham): [~/xplor/thesis]> head -30 cdih.dat
! file cdih_std.dat

! Backbone restraints are from Arnott fiber
! coordinates as reported by Altona (1982).
! Sequence length (per strand): 4
! There are 2 segments in this structure
! Segment #1 is named a and has sequence:ATGC
! Segment #2 is named b and has sequence:GCAT
!

!-----------------------------------------------
! Torsional angle alpha
! defined: O3'(n-1)-P-O5'-C5'
!-----------------------------------------------
! T2 of segment: a
assign (segid a and resid 1 and name O3') (segid a and resid 2 and name P  )
       (segid a and resid 2 and name O5') (segid a and resid 2 and name C5')
       1 -46.8  20  2

! G3 of segment: a
assign (segid a and resid 2 and name O3') (segid a and resid 3 and name P  )
       (segid a and resid 3 and name O5') (segid a and resid 3 and name C5')
       1 -46.8  20  2

! C4 of segment: a
assign (segid a and resid 3 and name O3') (segid a and resid 4 and name P  )
       (segid a and resid 4 and name O5') (segid a and resid 4 and name C5')
       1 -46.8  20  2
```

Cdih_make script

```
#! /usr/local/bin/perl
# cdih_make
# Written by Jon Lapham
# <lapham@tecate.chem.yale.edu>
# Last revised December 21 1995  -JPL

# set up hashes,defaults,etc...:
%angle_names = (
    "alpha",0,"beta",1,"gamma",2,"epsilon",3,"zeta",4,
    "chi",5,"nu0",6,"nu1",7,"nu2",8,"nu3",9,"nu4",10);
$segid=none;
$segment[1]=$segid;
$seq_num=0;

foreach (<>) {
    ($first) = (split)[0];

    # Is the first word a remark character?
    if (/^!/) {# print "Remark: $_";
        next;}

    # Is the first word a angle name?  If so, define the angle.
    if ($angle_names{$first} ne '') {

    ($angle_default[$angle_names{$first}],$flex_default[$angle_
names{$first}]) = (split)[1,2];
        # print "$first angle:
$angle_default[$angle_names{$first}]\n";
        next;}

    # Is the first word a segment name?  If so, define the
segid.
    if ($first eq "segment") {
        ++$segid_num;
        $segid = (split)[1];

        # $segment[] holds all the segment names for use later
        $segment[$segid_num] = $segid;
        $seq_num=0;
        next;}

    # Is the first word NOT a legit nucleotide letter?
    if (/^[^agcutAGCUT]/) {next};

    # Is the first nucleotide letter an a, g, c, u, or t?
    # if so, the the user wants to define new angles!
    if (/^[agcut]/) {

        # first, break the line into the nucleotide letter and
the new angles
        ($nuc,$temp) = split(/\s+/);

        # If the person puts the word "none" for the second
word
        # then there are no angle restrictions for that
nucleotide!
        if ($temp eq "none") {
            $list[$counter] = "$segid:$seq_num:$nuc:none:none";
            }
        else {
        (@temp) = split(/:|,/,$temp);
        (@angle) = @temp[0,2,4,6,8,10,12,14,16,18,20];
        (@flex) = @temp[1,3,5,7,9,11,13,15,17,19,21];

        # set new angle and flex to appropriate variable names

        # set Master list
        $list[$counter] = "$segid:$seq_num:$nuc:@angle:@flex";
        }
    }

    # The angles are of stardard form
    else {
        $nuc = $first;
        $list[$counter] =
"$segid:$seq_num:$nuc:@angle_default:@flex_default";
        }

    # All variables have been set, now we just have to print
them out!
    # print $list[$counter],"\n";
    ++$counter;
    ++$seq_num;
}

# Let's begin actually making the output file!!!
```

```
if ($segid_num eq "") {$segid_num = 1;}

print "! file cdih_std.dat \n\n";
print "! Backbone restraints are from Arnott fiber \n";
print "! coordinates as reported by Altona (1982).\n";
print "! Sequence length (per strand): $seq_num \n";
print "! There are $segid_num segments in this structure\n";

foreach $i (1 .. $segid_num) {
    print "! Segment #",$i," is named $segment[$i] and has
sequence:";
    foreach $j (($i-1)*$seq_num .. ($i*$seq_num-1)) {
        (@temp) = split(/:/,$list[$j]);
        print $temp[2];
    }
print "\n";
}

print "!\n";

&alpha;
&beta;
&gamma;
&epsilon;
&zeta;
&chi;
&nu0;
&nu1;
&nu2;
&nu3;
&nu4;

#####################################
# Define all the subroutines
#####################################
sub alpha {
print "\n!-------------------------------------------------\n";
print "! Torsional angle alpha\n";
print "! defined: O3'(n-1)-P-O5'-C5'\n";
print "!-------------------------------------------------\n";
$atom1="O3'"; $atom2="P  "; $atom3="O5'"; $atom4="C5'";
$alpha=1;
$position=alpha;
&assign;
```

```
$alpha=0;
}

sub beta {
print "\n!-------------------------------------------------\n";
print "! Torsion angle beta\n";
print "! defined: P-O5'-C5'-C4'\n";
print "!-------------------------------------------------\n";
$atom1="P  "; $atom2="O5'"; $atom3="C5'"; $atom4="C4'";
$position=beta;
&assign;
}

sub gamma {
print "\n!-------------------------------------------------
\n";
print "! Torsion angle gamma\n";
print "! Stem constrained to A-form => gamma= 45 +/- 30\n";
print "! defined: O5'-C5'-C4'-C3'\n";
print "!-------------------------------------------------\n";
$atom1="O5'"; $atom2="C5'"; $atom3="C4'"; $atom4="C3'";
$position=gamma;
&assign;
}

sub epsilon {
print "\n!-------------------------------------------------
\n";
print "! Torsion angle epsilon\n";
print "! defined: C4'-C3'-O3'-P(n+1)\n";
print "!-------------------------------------------------\n";
$atom1="C4'"; $atom2="C3'"; $atom3="O3'"; $atom4="P  ";
$epsilon=1;
$position=epsilon;
&assign;
$epsilon=0;
}

sub zeta {
print "\n!-------------------------------------------------
\n";
print "! Torsion angle zeta\n";
print "! defined: C3'-O3'-P(n+1)-O5'(n+1)\n";
print "!-------------------------------------------------\n";
$atom1="C4'"; $atom2="O3'"; $atom3="P  "; $atom4="O5'";
```

```
$zeta=1;
$position=zeta;
&assign;
$zeta=0;
}

sub chi {
print "\n!-----------------------------------------------\n";
print "! Torsion angle chi  \n";
print "!     200deg = _anti_; 60deg = _syn_\n";
print "! Defined (pur): O4'-C1'-N9-C4\n";
print "! Defined (pyr): O4'-C1'-N1-C2\n";
print "!-----------------------------------------------\n";
$chi=1;
$position=chi;
&assign;
$chi=0;
}

sub nu0 {
print "\n!-----------------------------------------------\n";
print "! Torsion angle nu zero\n";
print "! defined: C4'-O4'-C1'-C2'\n";
print "!-----------------------------------------------\n";
$atom1="C4'"; $atom2="O4'"; $atom3="C1'"; $atom4="C2'";
$position=nu0;
&assign;
}

sub nu1 {
print "\n!-----------------------------------------------\n";
print "! Torsion angle nu one\n";
print "! defined: O4'-C1'-C2'-C3'\n";
print "!-----------------------------------------------\n";
$atom1="O4'"; $atom2="C1'"; $atom3="C2'"; $atom4="C3'";
$position=nu1;
&assign;
}

sub nu2 {
print "\n!-----------------------------------------------\n";
print "! Torsion angle nu two\n";
print "! defined: C1'-C2'-C3'-C4'\n";
print "!-----------------------------------------------\n";
$atom1="C1'"; $atom2="C2'"; $atom3="C3'"; $atom4="C4'";
```

```
$position=nu2;
&assign;
}

sub nu3 {
print "\n!-----------------------------------------------\n";
print "! Torsion angle nu three\n";
print "! defined: C2'-C3'-C4'-O4'\n";
print "!-----------------------------------------------\n";
$atom1="C2'"; $atom2="C3'"; $atom3="C4'"; $atom4="O4'";
$position=nu3;
&assign;
}

sub nu4 {
print "\n!-----------------------------------------------\n";
print "! Torsion angle nu four\n";
print "! defined: C3'-C4'-O4'-C1'\n";
print "!-----------------------------------------------\n";
$atom1="C3'"; $atom2="C4'"; $atom3="O4'"; $atom4="C1'";
$position=nu4;
&assign;
}


# This is the big cahoona (sp?) which retrieves all info from
$list[] and prints it out
sub assign {

# Count up from one for each segment which exists
foreach $i (1 .. $segid_num) {
    # Call $segid the name of whichever segment name we are
working on now.
    # $segment[] holds all the segment names
    $segid = $segment[$i];

    # Count (0 .. x-1) for 1st segment, (x .. 2x-1) for 2nd
segment
    # This way we can use $list[] to retrieve all the info for
each nucleotide
    foreach $j (($i-1)*$seq_num .. ($i*$seq_num-1)) {
        (@temp) = split(/:/,$list[$j]);

        # Remember to skip over anything which has "none" for
angles!
```

```
       if ($temp[3] eq "none") { next;}

       $nuc_num = @temp[1]+1;
       $nuc = @temp[2];
       $temp_angle = @temp[3];
       $temp_flex = @temp[4];
       (@temp_angle) = split(/\s+/,$temp_angle);
       (@temp_flex) = split(/\s+/,$temp_flex);
       $angle = @temp_angle[$angle_names{$position}];
       $flex = @temp_flex[$angle_names{$position}];

       # if you are on alpha, you must subtract one from
$resid1
       if ($position eq "alpha") {$resid1=$nuc_num-1;
$resid2=$nuc_num; $resid3=$nuc_num; $resid4=$nuc_num;}

       # if you are on epsilon, you must add one onto $resid4
       elsif ($position eq "epsilon") {$resid1=$nuc_num;
$resid2=$nuc_num; $resid3=$nuc_num; $resid4=$nuc_num+1;}

       # if you are on zeta, you must add one to both $resid3
and $resid4
       elsif ($position eq "zeta") {$resid1=$nuc_num;
$resid2=$nuc_num; $resid3=$nuc_num+1; $resid4=$nuc_num+1;}

       # All others can use $nuc_num for $resid
       else {$resid1=$nuc_num; $resid2=$nuc_num;
$resid3=$nuc_num; $resid4=$nuc_num;}

       # Don't print out angles which extend 5' of the 5' end,
or 3' of the 3' end
       if (($resid1 eq "0") || ($resid3 > $seq_num) ||
($resid4 > $seq_num)) {next;}

       # Okay, now we have to take care of that DAMNABLE chi:
       if ($position eq "chi") {
           if ($nuc =~ /[AGag]/) {$atom1="O4'"; $atom2="C1'";
$atom3="N9 "; $atom4="C4 ";}
           else {$atom1="O4'"; $atom2="C1'"; $atom3="N1 ";
$atom4="C2 ";}
           }

       # For single strand with no segid defined
       if ($segid eq "none") {
           print "!",$nuc,$nuc_num,"\n";
           print "assign (resid $resid1 and name $atom1)";
           print " (resid $resid2 and name $atom2) \n";
           print "       (resid $resid3 and name $atom3)";
           print " (resid $resid4 and name $atom4) \n";
           print "       1 $angle  $flex  2\n";
           }

       # For multiple strands or single strand with segid
defined
       else {
           print "! ",$nuc,$nuc_num," of segment: $segid\n";
           print "assign (segid $segid and resid $resid1 and
name $atom1)";
           print " (segid $segid and resid $resid2 and name
$atom2) \n";
           print "       (segid $segid and resid $resid3 and
name $atom3)";
           print " (segid $segid and resid $resid4 and name
$atom4) \n";
           print "       1 $angle  $flex  2\n\n";
       }
    }
}
}
```

8.1.3 planar_make – create XPLOR planar restraint files

Dan Zimmer wrote the original version of this script. It creates XPLOR planar restraint files from an input seq file.

Syntax: **planar_make < seq_file > planar_restraint_file**

In this example, the DNA seq file from section 7.2.1 is used to create a planarity restraint file, and the first 30 lines of the output planar.dat file are shown.

```
bass (lapham): [~/xplor/thesis]> planar_make < atgc > planar.dat
bass (lapham): [~/xplor/thesis]> head -30 planar.dat
! file: planar.dat
! DNA
! Planar restraints to maintain base pair planarity
! This file was created by planar_make.pl
! Update October 22, 1996
! NOTE: $PSCALE MUST BE DEFINED WITHIN THIS FILE OR IN EACH PROTOCOL!
! OTHERWISE THE DEFAULT IS: 300kcal/mol/A^2

evaluate ($pscale = 50)

! A1-T4 Watson-Crick
!---------------------------------------------------------------------
group
selection= ((segid a and resid  1 and name n1) or (segid a and resid  1 and name n3) or
        (segid a and resid  1 and name c5) or (segid b and resid  4 and name n1) or
        (segid b and resid  4 and name n3) or (segid b and resid  4 and name c5))
weight = $pscale end

! T2-A3 Watson-Crick
!---------------------------------------------------------------------
group
selection= ((segid a and resid  2 and name n1) or (segid a and resid  2 and name n3) or
        (segid a and resid  2 and name c5) or (segid b and resid  3 and name n1) or
        (segid b and resid  3 and name n3) or (segid b and resid  3 and name c5))
weight = $pscale end

! G3-C2 Watson-Crick
!---------------------------------------------------------------------
group
selection= ((segid a and resid  3 and name n1) or (segid a and resid  3 and name n3) or
```

Planar_make script

```
#!/usr/local/bin/perl
# usage: planar_make seq_dna
# prints XPLOR planar constraint file for dna duplexes to
standard output

$segid_num=0;

foreach(<>){

        ($first) = (split)[0];

    # Is the first word a remark character?
    if (/^!/) {# print "Remark: $_";
        next;}

    # Is the first word a segment name?  If so, define the
segid.
    if ($first eq "segment") {
        ++$segid_num;
        $i=1;
        $segid = (split)[1];

        # $segment[] holds all the segment names for use later
        $segment[$segid_num] = $segid;
        $seq_num=0;
        next;}


    # Is the first word NOT a legit nucleotide letter?
    if (/^[^agcutAGCUT]/) {next};

    #
    if ($segid_num == 1) {
        $res[$i] = $first;
        $i++;
        $m=$i;
    }
}

$i=1;
```

```
print "! file: planar.dat\n";
print "! DNA\n";
print "! Planar restraints to maintain base pair planarity\n";
print "! This file was created by planar_make.pl\n";
print "! Update October 22, 1996\n";
print "! NOTE: \$PSCALE MUST BE DEFINED WITHIN THIS FILE OR IN
EACH PROTOCOL!\n";
print "! OTHERWISE THE DEFAULT IS: 300kcal/mol/A^2\n";
print "\n";
print "evaluate (\$pscale = 50)\n";
print "\n";

$i=1;
while ($res[$i]) {
    $j=$m-$i;
    if ($res[$i] =~ /[aA]/) {
        print "! A$i-T$j Watson-Crick\n";
        print "!-----------------------------------------------
--------------------\n";
        print "group\n";
        print "selection=\t((segid a and resid  $i and name n1)
or (segid a and resid  $i and name n3) or\n";
        print "\t(segid a and resid  $i and name c5) or (segid
b and resid  $j and name n1) or\n";
        print "\t(segid b and resid  $j and name n3) or (segid
b and resid  $j and name c5))\n";
        print "weight = \$pscale end\n";
    }
    elsif ($res[$i] =~ /[gG]/) {
        print "! G$i-C$j Watson-Crick\n";
        print "!-----------------------------------------------
--------------------\n";
        print "group\n";
        print "selection=\t((segid a and resid  $i and name n1)
or (segid a and resid  $i and name n3) or\n";
        print "\t(segid a and resid  $i and name c5) or (segid
b and resid  $j and name n1) or\n";
        print "\t(segid b and resid  $j and name n3) or (segid
b and resid  $j and name c5))\n";
        print "weight = \$pscale end\n";
    }
    elsif ($res[$i] =~ /[cC]/) {
        print "! C$i-G$j Watson-Crick\n";
        print "!-----------------------------------------------
--------------------\n";
```

```
    print "group\n";
    print "selection=\t((segid a and resid  $i and name n1)
or (segid a and resid  $i and name n3) or\n";
    print "\t(segid a and resid  $i and name c5) or (segid
b and resid  $j and name n1) or\n";
    print "\t(segid b and resid  $j and name n3) or (segid
b and resid  $j and name c5))\n";
    print "weight = \$pscale end\n";
  }
  elsif ($res[$i] =~ /[tT]/) {
    print "! T$i-A$j Watson-Crick\n";
    print "!---------------------------------------------
---------\n";
    print "group\n";
    print "selection=\t((segid a and resid  $i and name n1)
or (segid a and resid  $i and name n3) or\n";
    print "\t(segid a and resid  $i and name c5) or (segid
b and resid  $j and name n1) or\n";
    print "\t(segid b and resid  $j and name n3) or (segid
b and resid  $j and name c5))\n";
    print "weight = \$pscale end\n";
  }
  print "\n";
  $i++;
}
end
```

8.1.4 dm – measures the distances between protons in pdb files

Dave Schweisguth originally wrote this script. It measures the distances between protons in a pdb structure file.

Syntax: **dm < pdb_file > distance_output_file**

In this example, the distances between all the protons within 5Å of each other for the pdb file "dickerson.pdb" will be saved to a file, the first 20 lines of the file will then be examined.

```
bass (lapham): [~/xplor/thesis]> dm < dickerson.pdb > distances
bass (lapham): [~/xplor/thesis]> head -20 distances
A CYT 1 H1'     A CYT 1 H2''       2.372
A CYT 1 H1'     A CYT 1 H2'        3.032
A CYT 1 H1'     A CYT 1 H3'        3.933
A CYT 1 H1'     A CYT 1 H4'        3.646
A CYT 1 H1'     A CYT 1 H5'        4.461
A CYT 1 H1'     A CYT 1 H6         3.697
A CYT 1 H1'     A GUA 2 H1'        4.921
A CYT 1 H1'     A GUA 2 H2'        4.144
A CYT 1 H1'     A GUA 2 H3'        4.965
A CYT 1 H1'     A GUA 2 H4'        4.240
A CYT 1 H1'     A GUA 2 H5'        1.805
A CYT 1 H1'     A GUA 2 H5''       3.362
A CYT 1 H1'     A GUA 2 H8         2.828
A CYT 1 H2''    A CYT 1 H2'        1.758
A CYT 1 H2''    A CYT 1 H3'        2.703
A CYT 1 H2''    A CYT 1 H4'        4.095
A CYT 1 H2''    A CYT 1 H5'        4.964
A CYT 1 H2''    A CYT 1 H5''       4.939
A CYT 1 H2''    A CYT 1 H6         3.424
A CYT 1 H2''    A GUA 2 H2'        3.835
```

DM script

```
#!/usr/local/bin/perl

# dm ("Distance Matrix") v. 21 Nov 1995
# Dave Schweisguth <dcs@proton.chem.yale.edu>

($whatami = $0)    =~ s|.*/||;    # `basename $0`

$res_diff  = 1;
$d_diff    = 5;

while ($#ARGV > -1 && (($first, $rest) = ($ARGV[0] =~ /^-
(.)(.*)/))) {
    # Perl 5 lossage alert
    if ($first =~ /[dr]/) {   # Switches with arguments
        shift;
        $arg = $rest ne '' ? $rest : $ARGV[0] ne '' ? shift :
            &usage("$whatami: -$first requires an
argument.\n");
    } elsif ($rest ne '') {
        $ARGV[0] = "-$rest";
    } else {
        shift;
    }
    if ($first eq 'd')   { $d_diff = $arg; }
    elsif ($first eq 'r') { $res_diff = $arg; }
    elsif ($first eq 'u') { &usage(0); }
    else         { &usage("$whatami: -$first is not an
option.\n"); }
}

while(<>) {
    next unless ($atom_type, $res_type, $chain, $res_num, $x,
$y, $z, $seg) =
    /^(?:ATOM  |HETATM).{5} (.{5})(.{3}) (.)(.{4}).
(.{8})(.{8})(.{8}).{6}.{6}.{3}(.{4})/;
    # Column 17 ("alternate location") appended to columns 13-
16
    #     (atom type), mostly to compensate for bad Insight PDB
    # Columns 77-80 (element and charge fields) ignored
    foreach $i ($atom_type, $res_type, $chain, $res_num, $x,
$y, $z, $seg) {
    $i =~ s/^\s*(\S*)\s*/$1/;
    }
    foreach $i ($atom_type, $res_type, $res_num, $x, $y, $z) {
    # Chain and segment IDs not always present, so not required
        warn "$whatami: Bad ATOM record on line $.!\n" if $i
eq '';
    }
    next unless $atom_type =~ /^[12]?H/;
    push(@tag, $seg . ($seg ? ' ' : '') . $chain . ($chain ? '
' : '') .
    "$res_type $res_num $atom_type");
    push(@res_num, $res_num);
    push(@x, $x);
    push(@y, $y);
    push(@z, $z);
}

foreach $i (0 .. $#tag) {
    foreach $j (($i + 1) .. $#tag) {
    next if abs($res_num[$i] - $res_num[$j]) > $res_diff;
        $d = sqrt(($x[$i] - $x[$j]) ** 2 + ($y[$i] - $y[$j])
** 2 +
        ($z[$i] - $z[$j]) ** 2);
    printf("$tag[$i]\t$tag[$j]\t%8.3f\n", $d) unless $d >
$d_diff;
    }
}

sub usage {
    local ($message) = $_[0];

    warn $message if $message;
    warn <<EOP;
Usage: $whatami [-d #] [-r #] PDB-file
Options:
-d #    Print distances less than # (default 5)
-r #    Print distances between residue numbers differing by
less than #
        (default 1)
-u  This message
EOP
    exit !! $message;
}
```

8.1.5 noe_in – converts the output of dm to an XPLOR input format file

This is a simple script to convert the output of the dm script to a XPLOR readable restrain file. Jason Rife wrote the original version of the program.

Syntax: **noe_in < dm_output_file > distance_restraint_file**

In this example, the 'distances' file generated from the 'dm' script will be used to build an XPLOR distance restraint file. The first 20 line of the resultant restraint file will be examined.

```
bass (lapham): [~/xplor/thesis]> noe_in < distances > noe.dat
bass (lapham): [~/xplor/thesis]> head -20 noe.dat
assign (resid CYT and name 1)
         (resid A and name CYT)  1 0.1 0.1
assign (resid CYT and name 1)
         (resid A and name CYT)  1 0.1 0.1
assign (resid CYT and name 1)
         (resid A and name CYT)  1 0.1 0.1
assign (resid CYT and name 1)
         (resid A and name CYT)  1 0.1 0.1
assign (resid CYT and name 1)
         (resid A and name CYT)  1 0.1 0.1
assign (resid CYT and name 1)
         (resid A and name CYT)  1 0.1 0.1
assign (resid CYT and name 1)
         (resid A and name GUA)  2 0.1 0.1
assign (resid CYT and name 1)
         (resid A and name GUA)  2 0.1 0.1
assign (resid CYT and name 1)
         (resid A and name GUA)  2 0.1 0.1
assign (resid CYT and name 1)
         (resid A and name GUA)  2 0.1 0.1
```

The source code for the noe_in script:

```
#!/usr/local/bin/perl
# noe_in
# Creates noe constraint input file from 'dm' output file.
# Jason P. Rife 11/24/95
# Jon Lapham edit 3/14/96 to automatically remove HO2'
# usage: noe_in dm_file > out_file

while(<>) {
        ($junkA,$resIDA,$nameA,$junkB,$resIDB,$nameB,$dist) = split(/\s+/,$_);
    if (($nameA eq "H5'") && ($nameB eq "H5')) {next;}
    if (($nameA eq "HO2'") || ($nameB eq "HO2')) {next;}
    print "assign (resid $resIDA and name $nameA) \n  \t (resid $resIDB and name $nameB)
$dist 0.1 0.1 \n";
}
```

8.1.6 noe_hbond_make – builds an XPLOR hydrogen bonding restraint file

This script generates the hbond.dat restraint file which forces standard base pairing distances between two nucleotides. This is accomplished by defining the distances between a few heavy atoms as found in a standard Watson-Crick type base pair.

USAGE: **noe_hbond_make < seq_file > hbond.dat**

In the example below, the h-bond restraint file is generated from the input seq file as shown for the DNA ATGC in section 7.2.1. The first 2 base pairs of the resultant h-bonding restraint file will be examined.

```
bass (lapham): [~/xplor/thesis]> noe_hbond_make < atgc > noe_hbond.dat
bass (lapham): [~/xplor/thesis]> head -20 noe_hbond.dat
! base pairing constraint file
! created by noe_hbond_make.pl
!
! A1-T4 Watson-Crick (B-form DNA)
assign (segid A and resid  1 and name N1 )
       (segid B and resid  4 and name H3 )  1.92 0.20 0.20
assign (segid A and resid  1 and name N1 )
       (segid B and resid  4 and name N3 )  2.95 0.20 0.20
assign (segid A and resid  1 and name N6 )
       (segid B and resid  4 and name O4 )  2.81 0.20 0.20
assign (segid A and resid  1 and name H62)
       (segid B and resid  4 and name O4 )  1.78 0.20 0.20

! T2-A3 Watson-Crick (B-form DNA)
assign (segid A and resid  2 and name H3 )
       (segid B and resid  3 and name N1 )  1.92 0.20 0.20
assign (segid A and resid  2 and name N3 )
       (segid B and resid  3 and name N1 )  2.95 0.20 0.20
assign (segid A and resid  2 and name O4 )
       (segid B and resid  3 and name N6 )  2.81 0.20 0.20
assign (segid A and resid  2 and name O4 )
       (segid B and resid  3 and name H62)  1.78 0.20 0.20
```

Noe_hbond_make script

```
#! /usr/local/bin/perl
# noe_hbond_make.pl
# usage: noe_hbond_make.pl seq_dna
# prints to standard output xplor hbond contraints for duplex
DNA

$segid_num=0;

foreach(<>){

        ($first) = (split)[0];

    # Is the first word a remark character?
    if (/^!/) {# print "Remark: $_";
        next;}

    # Is the first word a segment name?  If so, define the
segid.
    if ($first eq "segment") {
        ++$segid_num;
        $i=1;
        $segid = (split)[1];
        $start_num = (split)[2];
        if ($start_num eq "") {$start_num =0;}
        $start_num[$segid_num]=$start_num;

        # $segment[] holds all the segment names for use later
        $segment[$segid_num] = $segid;
        $seq_num=0;
        next;}


    # Is the first word NOT a legit nucleotide letter?
    if (/^[^agcutAGCUT]/) {next};

    #
    if ($segid_num == 1) {
        $res[$i] = $first;
        $i++;
        $m=$i;
    }
```

```
}

print "! base pairing constraint file\n";
print "! created by noe_hbond_make.pl \n";
print "!\n";

$index=1;
while ($res[$index]) {
    $i=$index + $start_num[1];
    $j=$m-$i + $start_num[2];
    if ($res[$index] =~ /[aA]/) {
        print "! A$i-T$j Watson-Crick (B-form DNA)\n";
        print "assign (segid A and resid  $i and name N1 )
(segid B and resid  $j and name H3 )  1.92 0.20 0.20\n";
        print "assign (segid A and resid  $i and name N1 )
(segid B and resid  $j and name N3 )  2.95 0.20 0.20\n";
        print "assign (segid A and resid  $i and name N6 )
(segid B and resid  $j and name O4 )  2.81 0.20 0.20\n";
        print "assign (segid A and resid  $i and name H62)
(segid B and resid  $j and name O4 )  1.78 0.20 0.20\n";
    }
    elsif ($res[$index] =~ /[gG]/) {
        print "! G$i-C$j Watson-Crick (B-form DNA)\n";
        print "assign (segid A and resid  $i and name H1 )
(segid B and resid  $j and name N3 )  1.89 0.20 0.20\n";
        print "assign (segid A and resid  $i and name N1 )
(segid B and resid  $j and name N3 )  2.91 0.20 0.20\n";
        print "assign (segid A and resid  $i and name H22)
(segid B and resid  $j and name O2 )  1.98 0.20 0.20\n";
        print "assign (segid A and resid  $i and name N2 )
(segid B and resid  $j and name O2 )  3.01 0.20 0.20\n";
        print "assign (segid A and resid  $i and name O6 )
(segid B and resid  $j and name H42)  1.67 0.20 0.20\n";
        print "assign (segid A and resid  $i and name O6 )
(segid B and resid  $j and name N4 )  2.70 0.20 0.20\n";

    }
    elsif ($res[$index] =~ /[cC]/) {
        print "! C$i-G$j Watson-Crick (B-form DNA)\n";
        print "assign (segid A and resid  $i and name N3 )
(segid B and resid  $j and name H1 )  1.89 0.20 0.20\n";
        print "assign (segid A and resid  $i and name N3 )
(segid B and resid  $j and name N1 )  2.91 0.20 0.20\n";
        print "assign (segid A and resid  $i and name O2 )
(segid B and resid  $j and name H22)  1.98 0.20 0.20\n";
```

```
      print "assign (segid A and resid  $i and name O2 )
   (segid B and resid  $j and name N2 )  3.01 0.20 0.20\n";
      print "assign (segid A and resid  $i and name H42)
   (segid B and resid  $j and name O6 )  1.67 0.20 0.20\n";
      print "assign (segid A and resid  $i and name N4 )
   (segid B and resid  $j and name O6 )  2.70 0.20 0.20\n";
      }
      elsif ($res[$index] =~ /[tT]/) {
         print "! T$i-A$j Watson-Crick (B-form DNA)\n";
         print "assign (segid A and resid  $i and name H3 )
   (segid B and resid  $j and name N1 )  1.92 0.20 0.20\n";
      print "assign (segid A and resid  $i and name N3 )
   (segid B and resid  $j and name N1 )  2.95 0.20 0.20\n";
      print "assign (segid A and resid  $i and name O4 )
   (segid B and resid  $j and name N6 )  2.81 0.20 0.20\n";
      print "assign (segid A and resid  $i and name O4 )
   (segid B and resid  $j and name H62)  1.78 0.20 0.20\n";
      }
      $index++;
   }

   end
```

8.1.7 cdih_measure – measures the dihedral angles of nucleic acid pdb files

This script is useful for measuring the heavy atom torsion angles from a nucleic acid PDB file. For nucleic acids, 11 torsion angles completely describe the conformation of a nucleotide monomer, named α, β, γ, ε, ζ, X, nu0, nu1, nu2, nu3 and nu4 with definitions as given below

```
     alpha = O3'(n-1)-P-O5'-C5'          nu0 = C4'-O4'-C1'-
C2'
     beta = P-O5'-C5'-C4'                nu1 = O4'-C1'-C2'-
C3'
     gamma = O5'-C5'-C4'-C3'        nu2 = C1'-C2'-C3'-C4'
     epsilon = C4'-C3'-O3'-P(n+1)        nu3 = C2'-C3'-C4'-
O4'
     zeta = C3'-O3'-P(n+1)-O5'(n+1)      nu4 = C3'-C4'-O4'-
C1'
     chi(pur) = O4'-C1'-N9-C4
     chi(pyr) = O4'-C1'-N1-C2
```

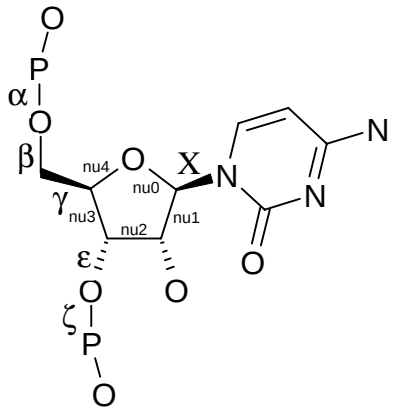

Figure 8. 4 Definition of torsion angles in nucleic acids

In order to perform this calculation, a general method for calculating torsion angles between two vectors that share a common third vector must be developed. Credit for this program must be given to discussions with Dan Zimmer and Charlie Schmuttenmaer.

This is a quick overview of how the torsion angles are calculated. Given that vectors **A** and **B** share are intersected by a common third vector **C**.

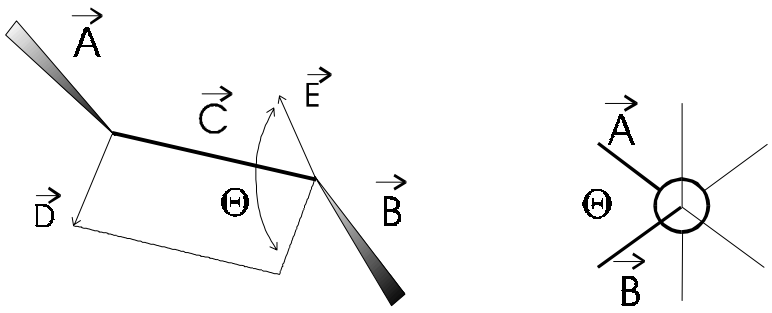

Figure 8. 5 Vector representation of the torsion angles

The cross product of **A** with **C** gives **D**, a vector orthogonal to both **A** and **C**. Likewise, the cross product of **B** with **C** gives **E**, a vector orthogonal to both **B** and **C**. Given that **A** and **B** both intersect **C**, the two planes that describe the possible positions of both **D** and **E** must be parallel. This requires that the angle that **D** and **E** make (as shown in the figure above with the two headed arrow) will truly represent θ, the torsion angle between **A** and **B**. The magnitude of the dot product between **D** and **E** gives the torsion angle.

$$\mathbf{A} \times \mathbf{C} = \mathbf{D}$$
$$\mathbf{B} \times \mathbf{C} = \mathbf{E}$$
$$\mathbf{D} \bullet \mathbf{E} = DE\cos\Theta = |D||E|\cos\Theta$$
$$\Theta = \arccos(|D||E|)$$

The next two pages show an example of the output from cdih_measure for the dickerson dodecamer 12 base pair DNA. The DNA was generated using the program Insight95 as standard B-form DNA. Notice that along with the individual torsion angles, the "P" (pseudo-rotation angle), numax and sugar pucker are calculated, as well. These sugar conformation calculations were an addition to the program by Dan Zimmer, thanks!

This example was run with the argument "define", which caused all the header information (definitions of angles, etc..) to be displayed. For more concise output, omit the word "define".

Cdih_measure example

```
bass (lapham): [~/xplor/thesis]> cdih_measure define dickerson.pdb > out
bass (lapham): [~/xplor/thesis]> cat out

    Name = Definition
   alpha = O3'(n-1)-P-O5'-C5'            nu0 = C4'-O4'-C1'-C2'
    beta = P-O5'-C5'-C4'                 nu1 = O4'-C1'-C2'-C3'
   gamma = O5'-C5'-C4'-C3'               nu2 = C1'-C2'-C3'-C4'
 epsilon = C4'-C3'-O3'-P(n+1)            nu3 = C2'-C3'-C4'-O4'
    zeta = C3'-O3'-P(n+1)-O5'(n+1)       nu4 = C3'-C4'-O4'-C1'
chi(pur) = O4'-C1'-N9-C4
chi(pyr) = O4'-C1'-N1-C2

Sugar pucker phase (P) = ( ( v4 + v1 ) - ( v3 + v0 ) ) /( 2 * v2 * (sin 36 + sin 72) )
Sugar pucker amplitude (vmax) = v2 / cos P

Sugar pucker definitions:
    -180 < P <= -144   C3'-exo
    -144 < P <= -108   C4'-endo
    -108 < P <=  -72   O4'-exo
     -72 < P <=  -36   C1'-endo
     -36 < P <=   0    C2'-exo
       0 < P <=      36    C3'-endo    A
      36 < P <=   72   C4'-exo
      72 < P <=  108   O4'-endo
     108 < P <=  144   C1'-exo
     144 < P <=  180   C2'-endo    B


Standard Values:
segid  num res  alpha    beta   gamma     eps    zeta     chi     nu0     nu1     nu2     nu3     nu4     P     numax   pucker
------ --- ---  ------ ------- ------- ------- ------- ------- ------- ------- ------- ------- ------- ------- ------- --------
A-RNA           -62    -180      47    -152     -74    -166       3     -25      37     -36      21      15      38      C3'-endo
B-DNA           -46.8  -146      36.4   155     -95.2   -97.8    -4.2    24.9   -34.9    33.3   -18.3  -168      36      C3'-exo
A-DNA           -83.9  -152.1    45.5    84.3   179.5   154.2     3.52  -26      37.1   -36      20.5   13       38      C3'-endo
poldA.poldT A   -44.91  127.93   37.15 -118.97 -169.75 -108.65  -47.91   56.25  -43.43   18.58   17.42  142.38  54.82   C1'-exo
poldA.poldT T   -41.39  135.58   37.46 -133.02 -159.65 -113.65  -41.49   51.40  -41.61   19.30   13.53  145.78  50.32   C2'-endo
```

```
filename: dickerson.pdb
segments:2  residues:12
```

segid	num	res	alpha	beta	gamma	eps	zeta	chi	nu0	nu1	nu2	nu3	nu4	P	numax	pucker
A	1	CYT	0.00	-146.02	36.36	154.98	-95.17	-97.94	-4.22	24.92	-34.92	33.25	-18.33	-168.20	35.67	C3'-exo
A	2	GUA	-46.84	-146.00	36.34	155.00	-95.16	-86.28	-4.23	24.93	-34.93	33.25	-18.32	-168.23	35.68	C3'-exo
A	3	CYT	-46.87	-146.03	36.40	155.00	-95.24	-97.96	-4.23	24.90	-34.87	33.19	-18.29	-168.24	35.62	C3'-exo
A	4	GUA	-46.81	-146.06	36.41	154.96	-95.16	-86.29	-4.25	24.96	-34.94	33.26	-18.32	-168.25	35.69	C3'-exo
A	5	ADE	-46.79	-146.00	36.34	155.01	-95.24	-86.23	-4.19	24.89	-34.89	33.23	-18.34	-168.17	35.65	C3'-exo
A	6	ADE	-46.80	-146.02	36.36	154.98	-95.17	-86.23	-4.22	24.92	-34.92	33.25	-18.33	-168.20	35.67	C3'-exo
A	7	THY	-46.84	-146.00	36.34	155.00	-95.16	-98.02	-4.23	24.93	-34.93	33.25	-18.32	-168.23	35.68	C3'-exo
A	8	THY	-46.87	-146.03	36.40	155.00	-95.24	-98.02	-4.23	24.90	-34.87	33.19	-18.29	-168.24	35.62	C3'-exo
A	9	CYT	-46.81	-146.06	36.41	154.96	-95.16	-97.96	-4.25	24.96	-34.94	33.26	-18.32	-168.25	35.69	C3'-exo
A	10	GUA	-46.79	-146.00	36.34	155.01	-95.24	-86.26	-4.19	24.89	-34.89	33.23	-18.34	-168.17	35.65	C3'-exo
A	11	CYT	-46.80	-146.02	36.36	154.98	-95.17	-97.94	-4.22	24.92	-34.92	33.25	-18.33	-168.20	35.67	C3'-exo
A	12	GUA	-46.84	-146.00	36.34	0.00	0.00	-86.28	-4.23	24.93	-34.93	33.25	-18.32	-168.23	35.68	C3'-exo
B	1	CYT	0.00	-146.00	36.34	155.01	-95.24	-97.95	-4.19	24.89	-34.89	33.23	-18.34	-168.17	35.65	C3'-exo
B	2	GUA	-46.80	-146.02	36.36	154.98	-95.17	-86.27	-4.22	24.92	-34.92	33.25	-18.33	-168.20	35.67	C3'-exo
B	3	CYT	-46.84	-146.00	36.34	155.00	-95.16	-97.96	-4.23	24.93	-34.93	33.25	-18.32	-168.23	35.68	C3'-exo
B	4	GUA	-46.87	-146.03	36.40	155.00	-95.24	-86.28	-4.23	24.90	-34.87	33.19	-18.29	-168.24	35.62	C3'-exo
B	5	ADE	-46.81	-146.06	36.41	154.96	-95.16	-86.26	-4.25	24.96	-34.94	33.26	-18.32	-168.25	35.69	C3'-exo
B	6	ADE	-46.79	-146.00	36.34	155.01	-95.24	-86.23	-4.19	24.89	-34.89	33.23	-18.34	-168.17	35.65	C3'-exo
B	7	THY	-46.80	-146.02	36.36	154.98	-95.17	-98.00	-4.22	24.92	-34.92	33.25	-18.33	-168.20	35.67	C3'-exo
B	8	THY	-46.84	-146.00	36.34	155.00	-95.16	-98.02	-4.23	24.93	-34.93	33.25	-18.32	-168.23	35.68	C3'-exo
B	9	CYT	-46.87	-146.03	36.40	155.00	-95.24	-97.96	-4.23	24.90	-34.87	33.19	-18.29	-168.24	35.62	C3'-exo
B	10	GUA	-46.81	-146.06	36.41	154.96	-95.16	-86.29	-4.25	24.96	-34.94	33.26	-18.32	-168.25	35.69	C3'-exo
B	11	CYT	-46.79	-146.00	36.34	155.01	-95.24	-97.95	-4.19	24.89	-34.89	33.23	-18.34	-168.17	35.65	C3'-exo
B	12	GUA	-46.80	-146.02	36.36	0.00	0.00	-86.27	-4.22	24.92	-34.92	33.25	-18.33	-168.20	35.67	C3'-exo

Cdih_measure script

```
#!/usr/local/bin/perl
# cdih_measure Author: Jon P Lapham
<lapham@tecate.chem.yale.edu>
# latest verion:
ftp//corona.chem.yale.edu/pub/scripts/cdih_measure
# Used to output tortion angles (heavy atom) from RNA/DNA .PDB
files.
# cdih_measure Last modified: 10-16-96  JPL for better non-
segid capabilities

%types = (
    "alpha", ["O3'","P","O5'","C5'"],
    "beta", ["P","O5'","C5'","C4'"],
    "gamma",["O5'","C5'","C4'","C3'"],
    "chi", ["O4'","C1'","xx","xx"],
    "eps", ["C4'","C3'","O3'","P"],
    "zeta", ["C3'","O3'","P","O5'"],
    "nu0", ["C4'","O4'","C1'","C2'"],
    "nu1", ["O4'","C1'","C2'","C3'"],
    "nu2", ["C1'","C2'","C3'","C4'"],
    "nu3", ["C2'","C3'","C4'","O4'"],
    "nu4", ["C3'","C4'","O4'","C1'"]
);

$seg_num=0;

if ($ARGV[0] eq "define") {
    print "\n    Name = Definition\n";
        print "   alpha = O3'(n-1)-P-O5'-C5'           nu0 =
C4'-O4'-C1'-C2'\n";
        print "    beta = P-O5'-C5'-C4'                nu1 =
O4'-C1'-C2'-C3'\n";
        print "   gamma = O5'-C5'-C4'-C3'              nu2 =
C1'-C2'-C3'-C4'\n";
        print " epsilon = C4'-C3'-O3'-P(n+1)           nu3 =
C2'-C3'-C4'-O4'\n";
        print "    zeta = C3'-O3'-P(n+1)-O5'(n+1)      nu4 =
C3'-C4'-O4'-C1'\n";
        print "chi(pur) = O4'-C1'-N9-C4\n";
        print "chi(pyr) = O4'-C1'-N1-C2\n";
    print "\nStandard Values:";
    print "
segid  num res  alpha    beta   gamma     eps    zeta     chi
nu0     nu1     nu2     nu3     nu4
------ --- ---  ------ ------- ------- ------- ------- -------
------- ------- ------- ------- -------
A-RNA           -62    -180      47    -152     -74    -166
3     -25      37     -36      21
B-DNA           -46.8  -146      36.4   155     -95.2   -97.8
-4.2    24.9   -34.9    33.3   -18.3
A-DNA           -83.9  -152.1    45.5    84.3   179.5   154.2
3.52  -26      37.1   -36      20.5\n\n";
    shift (@ARGV);
    }

$file=$ARGV[0];

foreach (<>) {

#    print;

    # Can't do the match and assignment in one step, because
we depend on the
    #   last assigned value of $num to be correct ... pity.

    next unless /^ATOM  .{5} (.{4}).(.{3}) .(.{4}).
(.{8})(.{8})(.{8})/;
    ($atom, $res, $num, $x, $y, $z) = ($1, $2, $3, $4, $5,
$6);

    # Match segid seperately, some pdb files don't use 'em
    if (/^ATOM  .{48}.{6}.{6} .{3}  (.{4})/) {
    ($segid) = ($1);
    # print "Segid: $segid\n";
    }

    # Trim whitespace from each field

    foreach $i ($atom, $res, $num, $x, $y, $z, $segid) {
    $i =~ s/^\s*([^\s]*)\s*$/$1/;
    }

    # Die if a field is empty which isn't allowed to be empty

    foreach $i ($atom, $res, $num, $x, $y, $z) {
    die "cdih_measure: Bad PDB record on input line ",
```

```
__LINE__, ":\n$_"
        if $i eq '';
    }


    if ($segid eq "") {
    $segid="";
    $seg_num=0;}
    else {
    if ($segid ne $segid[$seg_num]) {
        ++$seg_num;
        $segid[$seg_num]=$segid;
       }
    }

    # Set the main record keeping variables
    $res[$num]{$segid} = $res;
    $xyz[$num]{$atom,$segid} = [ $x, $y, $z ];
}

print "\nfilename: $file\nsegments:$seg_num  residues:$num\n";
print "segid  num res  alpha    beta   gamma     eps    zeta
chi     nu0     nu1     nu2     nu3     nu4\n";
print "------ --- ---  ------ ------- ------- ------- -------
------- ------- ------- ------- ------- -------\n";

# Not very elegant, but we have to loop through at least once
if ($seg_num eq 0) {$seg_num = 1;}
foreach $h (1 .. $seg_num) {
    $segid = $segid[$h];

foreach $i (1 .. $num) {
    next unless ($res[$i]{$segid});
    foreach $j (sort keys %types) {
    if ($j eq "chi") {
        ($atom_type1, $res_type1, $res_num1, $xyz1) =
        ($types{$j}[0], $res[$i]{$segid}, $i,
$xyz[$i]{$types{$j}[0],$segid});
        ($atom_type2, $res_type2, $res_num2, $xyz2) =
        ($types{$j}[1], $res[$i]{$segid}, $i,
$xyz[$i]{$types{$j}[1],$segid});
        if (($res[$i]{$segid} eq 'A') || ($res[$i]{$segid} eq
'G')) {
        ($atom_type3, $res_type3, $res_num3, $xyz3) =
            ("N9", $res[$i]{$segid}, $i,
```

```
$xyz[$i]{"N9",$segid});
        ($atom_type4, $res_type4, $res_num4, $xyz4) =
            ("C4", $res[$i]{$segid}, $i,
$xyz[$i]{"C4",$segid});
        } else {
        ($atom_type3, $res_type3, $res_num3, $xyz3) =
            ("N1", $res[$i]{$segid}, $i,
$xyz[$i]{"N1",$segid});
        ($atom_type4, $res_type4, $res_num4, $xyz4) =
            ("C2", $res[$i]{$segid}, $i,
$xyz[$i]{"C2",$segid});
        }
    } else {
        if ($j eq "alpha") {
        if ($i == 1) {
            $angle{'alpha'} = 0; next;
        } else {
            ($atom_type1, $res_type1, $res_num1, $xyz1) =
                ($types{$j}[0], $res[$i]{$segid}, $i-1,
$xyz[$i-1]{$types{$j}[0],$segid});
        }
        } else {
        ($atom_type1, $res_type1, $res_num1, $xyz1) =
            ($types{$j}[0], $res[$i]{$segid}, $i,
$xyz[$i]{$types{$j}[0],$segid});
        }

        ($atom_type2, $res_type2, $res_num2, $xyz2) =
        ($types{$j}[1], $res[$i]{$segid}, $i,
$xyz[$i]{$types{$j}[1],$segid});

        if ($j eq "zeta") {
        if ($res[$i+1]{$segid}) {
            ($atom_type3, $res_type3, $res_num3, $xyz3) =
            ($types{$j}[2], $res[$i+1]{$segid}, $i+1,
$xyz[$i+1]{$types{$j}[2],$segid});
        } else {
            $angle{'zeta'} = 0;
            next;
        }
        } else {
        ($atom_type3, $res_type3, $res_num3, $xyz3) =
            ($types{$j}[2], $res[$i], $i,
$xyz[$i]{$types{$j}[2],$segid});
        }
```

```
       if (($j eq "zeta") && $res[$i+1]{$segid}) {
       ($atom_type4, $res_type4, $res_num4, $xyz4) =
           ($types{$j}[3], $res[$i+1]{$segid}, $i+1,
$xyz[$i+1]{$types{$j}[3],$segid});
       } elsif ($j eq "eps") {
       if ($res[$i+1]{$segid}) {
            ($atom_type4, $res_type4, $res_num4, $xyz4) =
           ($types{$j}[3], $res[$i+1]{$segid}, $i+1,
$xyz[$i+1]{$types{$j}[3],$segid});
       } else {
           $angle{'eps'} = 0;
           next;
       }
       } else {
       ($atom_type4, $res_type4, $res_num4, $xyz4) =
           ($types{$j}[3], $res[$i]{$segid}, $i,
$xyz[$i]{$types{$j}[3],$segid});
       }
   }

   ($x1, $y1, $z1) = @$xyz1;
   ($x2, $y2, $z2) = @$xyz2;
   ($x3, $y3, $z3) = @$xyz3;
   ($x4, $y4, $z4) = @$xyz4;

   $r21x=$x2-$x1; $r21y=$y2-$y1; $r21z=$z2-$z1;
   $r23x=$x2-$x3; $r23y=$y2-$y3; $r23z=$z2-$z3;
   $r34x=$x3-$x4; $r34y=$y3-$y4; $r34z=$z3-$z4;
   $Ax=($r21y*$r23z)-($r21z*$r23y);
   $Ay=($r21z*$r23x)-($r21x*$r23z);
   $Az=($r21x*$r23y)-($r21y*$r23x);
   $Bx=($r34y*$r23z)-($r34z*$r23y);
   $By=($r34z*$r23x)-($r34x*$r23z);
   $Bz=($r34x*$r23y)-($r34y*$r23x);
   $magA=sqrt($Ax**2+$Ay**2+$Az**2);
   $magB=sqrt($Bx**2+$By**2+$Bz**2);
   $dot=($Ax*$Bx+$Ay*$By+$Az*$Bz);

   #Avoid a "divide by zero error"
   if ($magA*$magB ne 0) {
   $arccos=$dot/($magA*$magB);

   $bot=$arccos;
   $top=sqrt(1-$arccos**2);
```

```
   $theta=(180/3.1415927)*atan2($top,$bot);

   # new shit to calculate sign
   # calc theta2 between A and r34 because if <90, theta
   # should be positive, if >90 theta should be negative
   $mag34=sqrt($r34x**2+$r34y**2+$r34z**2);
   $dot2=($r34x*$Ax+$r34y*$Ay+$r34z*$Az);
   $arccos2=$dot2/($magA*$mag34);
   $theta2=(180/3.1415927)*atan2(sqrt(1-
$arccos2**2),$arccos2);
   if ($theta2 > 90) {$theta=$theta*-1;}

   # Set the output variables
   $angle{$j} = $theta;
   }
   else {
   $angle{$j} = 0;}
   }
   write;
}
print "\n";
}

################################################

format STDOUT =
@>>>>> @>> @>> @###.## @###.## @###.## @###.## @###.## @###.##
@###.## @###.## @###.## @###.## @###.##
$segid,$res_num2,$res_type2,$angle{'alpha'},$angle{'beta'},$an
gle{'gamma'},$angle{'eps'},$angle{'zeta'},$angle{'chi'},$angle
{'nu0'},$angle{'nu1'},$angle{'nu2'},$angle{'nu3'},$angle{'nu4'
}
.
```

8.2 Hydrodynamics

Calculation of the rotational and translational diffusion properties of regularly shaped hydrodynamic particles is presented in this section. These values are important for the evaluation of a number of NMR experiments. The rotational diffusion rate of a molecule (which can be expressed as a correlation time) is intimately related to the dipolar relaxation parameters for the molecule (see Chapter 5 of this thesis). The translational diffusion rate can be measured experimentally using the experiments presented in Chapter 4 of this thesis, and the ability to calculate a theoretic value is important in the interpretation of the results of the experiments.

Four hydrodynamic particle shapes are supported in this program; a sphere, a prolate ellipse, an oblate ellipse and a right cylinder. The equations for the calculations can be found in the references from Chapter 4 and 5. Below is an example of running the program for calculating first the rotational properties of a 12 mer DNA using the standard rise/base pair and diameter values in D2O at 25° C:

```
bass (lapham): [~]> hydro.pl
hydro.pl
    A program for simulating rotational and translational
    diffusion constants for nucleic acids from model
    hydrodynamic systems.

Would you like to enter the hydrodynamic parameters of a/b explicitly (y/[n])?n
Enter hydrodynamic diameter ([bdna], arna or angs): bdna
   using 20A diameter
Enter hydrodynamic rise/bp ([bdna], arna or angs): bdna
   using 3.4 rise/bp
Enter number of basepairs [12]: 12
   using 12 base pairs
Enter temperature (celcius)[25]: 25
   using 25 C
Enter viscosity ([d2o], h2o or user_specified): d2o
   using d2o for viscosity
Translational or rotational calculations ([trans] or rot): rot
   calculating rotational values
      units of Dr are (s-1)

 T  #bp Nu        Dr_s     Dr_pe_a  Dr_pe_b  Dr_cr_a  Dr_cr_b
 25  12 1.097e-03 1.76e+07 1.41e+08 2.03e+07 5.92e+07 2.58e+07
```

The results from this calculation are the the rotaional diffusion rate for the DNA is $1.76x10^7$ for the spherical model, $1.41x10^8$ and $2.03x10^7$ for the two axis of the prolate ellisoid model and $5.92x10^7$ and $2.58x10^7$ for the two axis of the cylindrical rod model. All the rates are, naurally, in units of s^{-1}. To convert the rotational diffusion rates (D_r) to correlation times (t_c), the equation is:

$$t_c = \frac{1}{6 \cdot D_r}$$

For example, the correlation time of the long axis of the DNA using the cylindrical rod model is:

$$t_l = \frac{1}{6 \cdot 5.92x10^7 s^{-1}} = 2.81ns$$

Hydro.pl script

```
#! /usr/local/bin/perl
# hydro.pl
# calculate translational and rotational diffusion constants
# using a variety of hydrodynamic shape models, such as a
# sphere, ellipse and cylinder.

############################################################
# main_program
############################################################

&opening_message;
&preliminaries;
&collect_info;
&header;

#### Begin main loops for bps and temperature
# They may only be looped once if they were explicitly
specified
foreach $bp ($bp_start .. $bp_stop) {
foreach $temp_c ($temp_start .. $temp_stop) {

# Calculate temperature in K, viscosity and a/b
&temp_viscosity_a_b_calc;

#### Calculations using the shape models
&sphere;
&prolate_ellipse;
&oblate_ellipse;
&symmetric_cylinder;

&print_data;

}
}

exit;

############################################################
# opening_message
#  - This is where the user is given short instructions
############################################################
sub opening_message {

print "\nhydro.pl\n";
print "    A program for simulating rotational and
translational\n";
print "    diffusion constants for nucleic acids from model
\n";
print "    hydrodynamic systems.\n\n";

}

############################################################
# preliminaries
#  - This is where constants are defined, such as
#    pi and planck's constant.
#  - Whenever possible, prefered units are meters
############################################################
sub preliminaries {

# Sometimes I use this for error reporting:
($whatami = $0) =~ s|.*/||;                    # `basename
$0`

# Definition of constants
$pi = 3.1415927;
# Planck's constant = k
# units = 1.38e-23 J K-1
#       = 1.38e-23 Kg m2 s-2 K-1
#       = 1.38e-19 Kg cm2 s-2 K-1
$k = 1.38066e-23;

}

############################################################
# collect_info
#  - This is where we read in the user input data, such as
#    length, diameter, temp, # bps,
############################################################
sub collect_info {

# Enter hydrodynamic parameters
print "Would you like to enter the hydrodynamic parameters of
a/b explicitly (y/[n])?";
    $pregunta = <STDIN>; chop $pregunta;
if ( ($pregunta eq "y") ) {
    print "Enter value for 2*a: "; $h_length = <STDIN>; chop
$h_length;
```

```
    print "Enter value for 2*b: "; $h_diameter = <STDIN>; chop
$h_diameter;
}
else {
print "Enter hydrodynamic diameter ([bdna], arna or angs): ";
    $h_diameter = <STDIN>; chop $h_diameter;
if (($h_diameter eq "bdna") || ($h_diameter eq "")) {
    $h_diameter = 20;
}
elsif ($h_diameter eq "arna") {
    $h_diameter = 24;
}
# Confirm diameter value
print "   using ",$h_diameter,"A diameter\n";

print "Enter hydrodynamic rise/bp ([bdna], arna or angs): ";
    $h_rpb = <STDIN>; chop $h_rpb;
if (($h_rpb eq "bdna") || ($h_rpb eq "")) {
    $h_rpb = 3.4;
}
elsif ($h_rpb eq "arna") {
    $h_rpb = 2.6;
}
# Confirm rise/bp value
print "   using $h_rpb rise/bp\n";

print "Enter number of basepairs [12]: ";
    $nbp=<STDIN>; chop $nbp;
if ($nbp eq "") { $nbp = 12; }
print "   using $nbp base pairs\n";
}

print "Enter temperature (celcius)[25]: ";
    $temp=<STDIN>; chop $temp;
if ($temp eq "") { $temp = 25; }
print "   using $temp C\n";

print "Enter viscosity ([d2o], h2o or user_specified): ";
    $nu_type=<STDIN>; chop $nu_type;
if ($nu_type eq "") { $nu_type = "d2o"; }
print "   using $nu_type for viscosity\n";

print "Translational or rotational calculations ([trans] or
rot): ";
    $calc_type = <STDIN>; chop $calc_type;
```

```
if (($calc_type eq "trans") || ($calc_type eq "")) {
    print "   calculating translational values\n";
    print "      units of Dt are (m2 s-1)\n";
}
else {
    print "   calculating rotational values\n";
    print "      units of Dr are (s-1)\n";
}

# Define the start and stop temperatures (or only one for a
specific temp)
if ($temp eq "all") {
    # Geesh, there must be an easier way...
    $temp_start = 1;
    $temp_stop = 40;
}
else {
    $temp_start = $temp;
    $temp_stop = $temp;
}

#### Define the start and stop bps (or only one for a specific
nbp or a)
# this loop will not be needed if a is defined, so use a dummy
value of 1
if ($nbp eq "all") {
    $bp_start = 1;
    $bp_stop = 40;
}
elsif ($a eq "") {
    $bp_start = $nbp;
    $bp_stop = $nbp;
}
else {
    $bp_start = 1;
    $bp_stop = 1;
}

}

############################################################
# header
#  - This is the header, specific for either translational
#    or rotational calculations
############################################################
```

```
sub header {

if (($calc_type eq "trans") || ($calc_type eq "")) {
# Print translational header
print "\n T  #bp Nu        Dt_s     Dt_pe     Dt_oe
Dt_cr\n";
}
else{
# Print rotational header
print "\n T  #bp Nu        Dr_s     Dr_pe_a  Dr_pe_b  Dr_cr_a
Dr_cr_b\n";
}

}

############################################################
# temp_viscosity_a_b_calc
############################################################
sub temp_viscosity_a_b_calc {

# Convert temp_c to Kelvin
$temp_k = $temp_c + 273.15;

# Calculate viscosity
# units: nu = kg m-1 s-1
if ($nu_type eq "d2o") {
    # D2O calculation
    $nu=10**(-4.2911+(-164.97/(174.24-$temp_k)));
}
elsif ($nu_type eq "h2o") {
    # H2O calculation
    $nu=10**(-4.5318+(-220.57/(149.39-$temp_k)));
}
else {
    # User specified viscosity
    $nu=$nu_type;
}

# Calculate a and b
# units a = b = meters
if ($h_length eq "") {
    # Must calculate a using rise/bp values
    $a = (($bp * $h_rpb) * 1e-10)/2;  # Remember, a is half of
the length!
}
```

```
else {
    # h_length was specified
    $a = $h_length*(1e-10)/2;
}
$b = $h_diameter*(1e-10)/2;       # Remember, b is half of the
diameter!

# Calculate length
$L = $a*2;

}

############################################################
# print_data
############################################################
sub print_data {

if (($calc_type eq "trans") || ($calc_type eq "")) {
    # Print translational data
    printf ("%3.0d %3.0d %1.3e %1.2e %1.3e %1.3e %1.3e\n",
    $temp_c, $bp, $nu, $Dt_s, $Dt_pe, $Dt_oe, $Dt_cr);
}
else {
    # Print rotational data
    printf ("%3.0d %3.0d %1.3e %1.2e %1.2e %1.2e %1.2e
%1.2e\n",
    $temp_c, $bp, $nu, $Dr_s, $Dr_pe_a, $Dr_pe_b, $Dr_cr_a,
$Dr_cr_b);
}

}

############################################################
# sphere
############################################################
sub sphere {

# For a sphere, a=b=R...
$R = $a;
# Translational calculations
$Ft_s = 6 * $pi * $nu * $R;
$Dt_s = $k * $temp_k / $Ft_s;

# Rotational calculations
$Fr_s = 8 * $pi * $nu * $R**3;
```

```
$Dr_s = $k * $temp_k / $Fr_s;
}

############################################################
# prolate_ellipse
# a must be greater than b
############################################################
sub prolate_ellipse {

# Axial ratio
$p = $b/$a;

# Translational calculations
# Re is the spherical equivalent radius
$temp_ab = ($a * ($b**2))**(1/3);
$temp1 = $temp_ab * (1-($p**2))**0.5;
$temp2 = ( $p**(2/3) ) * log( (1 + (1-($p**2))**0.5) / $p );
$Re =  $temp1 / $temp2;
$Ft_pe = 6 * $pi * $nu * $Re;
$Dt_pe = $k * $temp_k / $Ft_pe;

# Rotational calculations
$temp_1 = 1 + (1-($p**2))**(0.5);
$S = 2*((1-$p**2)**(-0.5)) * log( $temp_1/$p );
$Re_a = $temp_ab * 4 * (1-($p**2)) / ( 3 * (2-(($p**2)*$S)) );
$Re_b = $temp_ab * 4 * (1-($p**4)) / ( 3 * ($p**2) * ($S*(2-
($p**2))-2) );

$Fr_pe_a = 8 * $pi * $nu * ($Re_a**3);
$Fr_pe_b = 8 * $pi * $nu * ($Re_b**3);

$Dr_pe_a = $k * $temp_k / $Fr_pe_a;
$Dr_pe_b = $k * $temp_k / $Fr_pe_b;

if ($p > 1) {
    # $a is bigger than $b, i.e.:DNA shorter than 6 bp...
    # This keeps us from getting 'nan' (not a number) outputs
    $Ft_pe = 0; $Fr_pe_a = 0; $Fr_pe_b = 0; $Dt_pe = 0;
}

}

############################################################
# oblate_ellipse
############################################################
```

```
sub oblate_ellipse {

# Axial ratio
$p = $a/$b;

# Translational calculations
$temp_ab = ($a*($b**2))**(1/3);
$temp_1 = (($p**2)-1)**0.5;
$temp_2 = ($p**(2/3)) * atan2( (($p**2)-1)**0.5 , 1 );
$Re = $temp_ab * $temp_1 / $temp_2;
$Ft_oe = 6 * $pi * $nu * $Re;
$Dt_oe = $k * $temp_k / $Ft_oe;

# Rotational calculations
$S = 2*(1-$p**2)**(-0.5) * log( (1+(1-$p)**(0.5))/$p );
$Re_a = 4 * (1-$p**2) / ( 3 * (2-($p**2)*$S) );
$Re_b = $temp_ab * 4 * (1-$p**4) / ( 3 * ($p**2) * ($S*(2-
($p**2))-2) );
$Fr_oe_a = 8 * $pi * $nu * ($Re_a**3);
$Fr_oe_b = 8 * $pi * $nu * ($Re_b**3);

if ($p < 1) {
    # $a is bigger than $b, i.e.:DNA shorter than 6 bp...
    # This keeps us from getting 'nan' (not a number) outputs
    $Ft_oe = 0; $Dt_oe = 0;
}

}

############################################################
# symmetric_cylinder
############################################################
sub symmetric_cylinder {

# Axial ratio
$p = $a/$b;

# Translational calculations
# ft = (6*pi*viscosity of solution)* [(L/2) / (ln P + 0.312 +
0.565/P - 0.100/P^2)]
$temp1 = ($L/2);
$temp2 = ((log $p) + 0.312 + (0.565/$p) - (0.100/($p**2)));
$Re = $temp1 / $temp2;
$Ft_cr = 6 * $pi * $nu * $Re;
$Dt_cr = $k * $temp_k / $Ft_cr;
```

```
# Negative numbers are meaningless!!!!
if ($Dt_cr < 0) {
    $Dt_cr = 0; $Ft_cr = 0;
}

# Rotational calculations
$delta = -0.662 + 0.917/$p - 0.050/($p**2);

$Re = (3/(2*($p**2)))**(1/3) * ($L/2);
$Fr_cr_a = 0.64*(1.0 + 0.677/$p - 0.183/$p**2);
$Fr_cr_b = 2*($p**2) / ( 9*(log($p) + $delta ) );
$fr_cr_a = 8 * $pi * $nu * ($Re**3) * $Fr_cr_a;
$fr_cr_b = 8 * $pi * $nu * ($Re**3) * $Fr_cr_b;

$Dr_cr_a = $k * $temp_k / $fr_cr_a;
$Dr_cr_b = $k * $temp_k / $fr_cr_b;

# This is Pecora's way, it gives the same result...
# $Dr_cr_b = 3*$k*$temp_k*(log($p) + $delta) / (
$pi*$nu*($L**3) );

}
```

8.3 Moment of Inertia

Many of the calculations presented in this thesis require knowledge of the "principal axis of rotation" for a given molecule. For instance, the definition of the spectral density function for anisotropic rotation as formulated in section 5.5.3, has a β term, defined as the angle the ij atom pair makes with respect to the principal axis.

The rigorously correct method for determining the principal axis of rotation for a molecule in a solvent, would be to exactly determine the rotational diffusion tensor. However, this calculation is extremely difficult to perform, as it requires knowledge of the frictional coefficient. Approximation methods have been developed for certain hydrodynamic "regular" shapes, such as spheres, ellipses and cylinders (see the previous section, 7.3). But, what if you, the biomolecular spectroscopist, have a uniquely unusual shaped biomolecule and you want a rough estimate of the propensity of the structure to rotate in an anisotropic manner? And you want a quantitative method for determining the principal axis of rotation (read: so computer programs can take over the process).

The only method I have found for accomplishing these goals is to determine the "moments of inertia" for the molecule. This was derived from discussions with Profs. Kurt Zilm and Charlie Schmuttenmaer and the mathematics comes directly from the text "Classical Dynamics of Particles and Systems". The theory is developed in chapter 5 in section 5.8.2, if you are interested.

This is a program that calculates the inertia tensor of an arbitrary molecular structure and returns information related to the moments of inertia for that molecule.

8.3.1 Examples

Below are a few examples of how to use this program in everyday life, and how to incorporate it into other programs. The text shown below is the actual output from the program.

Running the program 'principal_axis' with no command line arguments prints out a short usage listing:

```
bass (lapham): [~/xplor/dickerson]> principal_axis
USAGE:
principle_axis <-options> <pdb file>
options:
    -segid  : Calculate segments seperately
    -nomass : Use mass 1 for every atom
    -midas  : Automatically start midas
    -report : Print a report to STDOUT
    -base   : Use only nucleic acid base heavy atoms
    -range  : Prompt for valid residues
    -xy     : Print small X and Y axis
    -short  : Print only the 3 principle axis XYZ components
```

The simplest way to use the program (and probably all most people will need) is to use the '-short' qualifier:

```
bass (lapham): [~/xplor/dickerson]> principal_axis -short dick_b.pdb
0.159361767996386 0.0325789844992612 0.986682540977625
```

The numbers that are returned above are the normalized unit length vector that lies parallel to the principal axis of the molecule (dick_b.pdb in this case).

The most information rich method of using the program is to use the '-report' qualifier. This prints to standard output a report that includes the actual numbers used in the initial inertia tensor matrix, the eigenvalues and eigenvectors of the inertia tensor, the molecular center of mass, the principal axis vector components, the number of segments (XPLOR definition of segments) and number of atoms used in the calculation.

```
bass (lapham): [~/xplor/dickerson]> principal_axis -report dick_b.pdb

============================================================
Calculation for pdbfile:dick_b.pdb segment: all

Starting matrix:
               x                y                z
```

```
       ---------          ---------          ---------
593543.056251672  3262.04898805153  -72434.171595150
3262.04898805153  577973.569530177  -14798.775561448
-72434.171595150  -14798.775561448  157924.089998264

Diagonalized matrix (eigenvalues):
               x                  y                  z
       ---------          ---------          ---------
  6.06386485E+05                  0                  0
               0     5.77317817E+05                  0
               0                  0     1.45736414E+05

Transformation matrix (eigenvectors):
               x                  y                  z
       ---------          ---------          ---------
  9.67070506E-01    -1.98440075E-01     1.59361768E-01
  1.95747721E-01     9.80112973E-01     3.25789845E-02
 -1.62657512E-01    -3.11472103E-04     9.86682541E-01

The resultant MAJOR principle axis has:
X components: 1.59361768E-01
Y components: 3.25789845E-02
Z components: 9.86682541E-01
with:
actual X:7.87716043957749
actual Y:2.42422749530196
actual Z:68.6825969478827
and is centered at:
X component: -0.0909279604225049
Y component: 0.795278270301956
Z component: 19.3484698978827
There are 1 segments, segid_num is 0
with 380 valid atoms in this segment
============================================================
```

Notice the three eigenvalue numbers above, X = 6.06×10^5, Y = 5.77×10^5 and Z = 1.46×10^5. These numbers are proportional to the actual "moments of inertia" about the principal axis vector, the smaller the number, the smaller the inertial moment about that axis. Thus, we can learn much about this molecule based on these numbers. If the three moments of inertia were the same, the molecule is described as a "spherical top". This means that the molecule is described by a single moment of inertia. Examples would be a sphere, a perfect cube or any other shape symmetric about all three axis.

In the example we used above, the Z component of the inertial moment is smaller than the other two. This is called a "symmetrical top" and it means that the shape is described by two moments of inertia, one about the principal axis and one about each of the other two axis.

The '-range' qualifier allows one to input a specific range of nucleic acid residues to use in the calculation. In the example below, only residue numbers 1, 2, 3, 4 and 5 will be used in the calculation:

```
bass (lapham): [~/xplor/dickerson]> principal_axis -range -short dick_b.pdb
Enter valid residue numbers, separated by spaces: 1 2 3 4 5

okay, calculating only for residue(s) 1 2 3 4 5
-0.469957055070988 0.501848901075805 0.726146023109685
```

The '-nomass' qualifier causes the program to arbitrarily weight all atoms with an atomic mass unit of 1. Normally, the correct mass of the atoms is used, consequently the heavier atoms (C, O, N, etc..) are weighted much more heavily (as they should be) in the calculation than the hydrogens.

Two nucleic acid specific qualifier were created. The '-base' qualifier will force the program to use only the base atoms in the calculation. This is useful for short segments of nucleic acid when you want to obtain a principal axis vector that runs through the center of the helix.

The '-segid' qualifier will perform the calculations on each segment (using the XPLOR definition of segid) separately.

A number of additional qualifiers were created for visualizing the results. The '-midas' qualifier causes midas to launch, graphically displaying the pdb file overlayed with either just the principal axis vector, or (with the '–xy' qualifier) with all three axis of the moments of inertia, as shown below in figure 7.5.

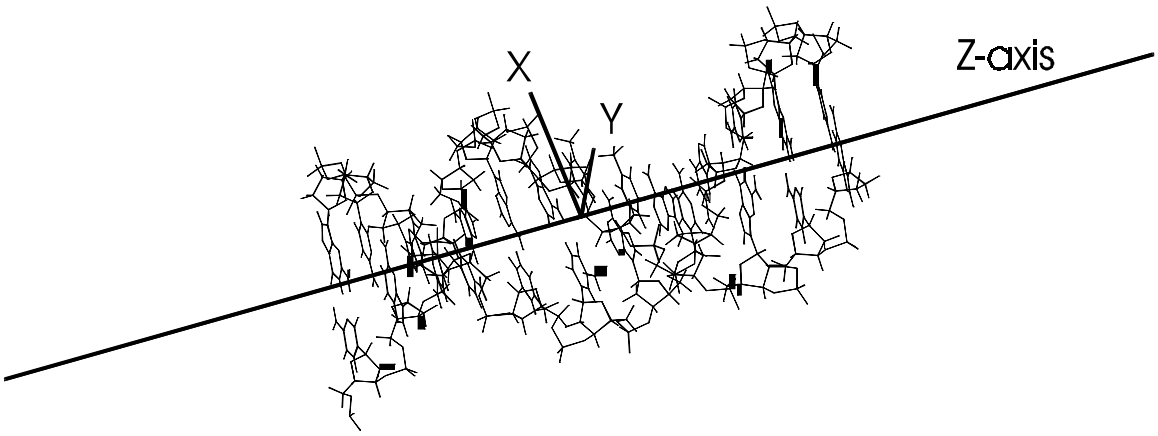

Figure 8. 6 principal_axis -xy -midas dickerson.pdb

7.4.2 principal_axis – perl code

```
#! /usr/local/bin/perl
# Written by Jon Lapham
# lapham@tecate.chem.yale.edu
# last edited 5/23/97

############################################################
# Preliminaries
############################################################

($whatami = $0) =~ s|.*/||;          # `basename $0`
$analysis_type="all";
$report="false";
$use_mass="true";
$base="false";
$launch_midas="false";
$range="false";
$bogus_segid="";
$xy="false";
$short="false";

# Parse incoming options, as long
# as the first argument begins with a minus sign:
while (($first,$rest) = ($ARGV[0] =~ /^-(\S+)(.*)/)) {
    if ($first eq "segid"){ $analysis_type = "segid"; }
    elsif ($first eq "nomass")    { $use_mass = "false"; }
    elsif ($first eq "midas") { $launch_midas = "true";}
    elsif ($first eq "report")    { $report = "true";}
    elsif ($first eq "base")  { $base = "true";}
    elsif ($first eq "range") { $range = "true";}
    elsif ($first eq "xy"){ $xy = "true";}
    elsif ($first eq "short") { $short = "true";}
    else { die "$whatami: $first is not an option.\n"; }
    if ($rest ne '') {
    $ARGV[0] = "-$rest";
    }
    else {shift;}
}

if ($ARGV[0] eq "") {
    print "USAGE:\n";
    print "principle_axis <-options> <pdb file>\n";
    print "options: \n";
    print "    -segid  : Calculate segments seperately\n";
    print "    -nomass : Use mass 1 for every atom\n";
    print "    -midas  : Automatically start midas\n";
    print "    -report : Print a report to STDOUT\n";
    print "    -base   : Use only nucleic acid base heavy
atoms\n";
    print "    -range  : Prompt for valid residues\n";
    print "    -xy     : Print small X and Y axis\n";
    print "    -short  : Print only the 3 principle axis XYZ
components\n";
    exit;
}

$pdb_file = $ARGV[0]; shift;

############################################################
# Main program
############################################################

if ($range eq "true") {
    print "Enter valid residue numbers, separated by spaces:
";
    $valid_res = <STDIN>; chop $valid_res;
    @valid_res = split(/ /,$valid_res);
    print "\nokay, calculating only for residue(s)
@valid_res\n";
}

&pdb_file_read;

# Analyze every atom in the pdb file in one fell swoop:
if ($analysis_type eq "all") {
    $segid_num=0;
    &reset_inertia_prep;
    foreach $segid_num (@segid) {
    &center_mass_calc;
    }
    $segid_num=0;
    foreach $segid_num (@segid) {
    &inertia_matrix_prep;
    }
    &inertia_calc;
    &vector_calc;
    if ($report eq "true") {&report;}
```

```
    if ($short eq "true") {&report_short;}
    $ext="vector";
    &write_vector_pdb_file;
    if ($launch_midas eq "true") {
    print "Launching midas...\n";
    `midas $pdb_file $pdb_file.vector &`;
    }
    exit;
}

# Analyze each segment in the pdb file separately:
elsif ($analysis_type eq "segid") {
    $segid_num=0;
    foreach $segid_num (@segid) {
    &reset_inertia_prep;
    &center_mass_calc;
    &inertia_matrix_prep;
    &inertia_calc;
        &vector_calc;
        if ($report eq "true") {&report;}
        if ($short eq "true") {&report_short;}
    $ext="vector.$segid_num";
        &write_vector_pdb_file;
    }
    if ($launch_midas eq "true") {
    print "Launching midas...\n";
    `midas $pdb_file $pdb_file.vector.* &`;
    }
    exit;
}

# I don't know how to do this analysis:
else {
    print "ERROR: Unknown analysis type\n";
    exit;
}

exit;


############################################################
# pdb_file_read
############################################################

sub pdb_file_read {
```

```
# dummy name for first segid if not defined:
$segid=0;
push(@segid, $segid);
$segid_num=0;
$atom_num[$segid_num]=0;

open(IN, $pdb_file) || die "Couldn't open $pdb_file for
reading \n";
while(<IN>) {
    # Some pdb files use TER to separate segments
    if (/^TER/) {
        # Check to see if this is the first line:
        if ($atom_num[$segid_num] == 0) {next;}

        # If bogus_segid exists, we are using made up segment
names
        if ($bogus_segid eq "true") {
            ++$segid_num;
            $segid[$segid_num] = ++$segid;
        $atom_num[$segid_num]=0;
            next;
        }
        # We are using real segids defined in the pdb file
        else {
            next;
        }

    }

    ########################################
    # Check to make sure each line meets the requirements
    # to be a valid entry
    ########################################
    next unless /^ATOM  .{5} (.{4}).(.{3}) .(.{4}).
(.{8})(.{8})(.{8})/;
    ($atom, $res, $num, $x, $y, $z) = ($1, $2, $3, $4, $5,
$6);

    # Trim whitespace from each field
    foreach $i ($atom, $res, $num, $x, $y, $z) {
        $i =~ s/^\s*([^\s]*)\s*$/$1/;
    }

    # Only allow base carbons/nitrogens through
    if ($base eq "true") {
```

```
    next unless ($atom =~ /C2|C4|C5|C6|C8|N1|N3|N7|N9/);
    }

    if ($range eq "true") {
    next unless (grep(/$num/,@valid_res));
    }

    # Match segid seperately, some pdb files don't use 'em
    if ($bogus_segid eq "") {
        if (/^ATOM  .{48}.{6}.{6} .{3}  (.{4})/) {
            $temp=$1;
            if ($temp eq "    ") {
        $bogus_segid = "true";
        $segid_num=0;
        $segid[$segid_num] = $segid;
        $segid[$segid_num] = $segid;
        $atom_num[$segid_num] = 0;
        }
            # Okay, we have a real segid name:
            else {
                $segid = $1;
                $segid =~ s/^\s*([^\s]*)\s*$/$1/;
                if ($segid[$segid_num] ne $segid) {
                    # Check to see if it segid_num has been
set yet:
                    if ($segid_num eq "0") {
                        $segid_num=0;
                        $segid[$segid_num] = $segid;
                        $atom_num[$segid_num] = 0;
                    }
                        else {
                        ++$segid_num;
                        $segid[$segid_num] = $segid;
                        $atom_num[$segid_num] = 0;
                    }
                }
            }
        }
        else {
        $bogus_segid = "true";
        $segid_num=0;
        $segid[$segid_num] = $segid;
        $segid[$segid_num] = $segid;
            $atom_num[$segid_num] = 0;
    }
```

```
    }

    # print;

    # Die if a field is empty which isn't allowed to be empty
    foreach $i ($atom, $res, $num, $x, $y, $z) {
    die "pdb_file_read subroutine: Bad PDB record on input
line ", __LINE__, ":\n$_"
        if $i eq '';
    }

    # print "$res $num $atom $x $y $z\n";
    # print "saved into segid_num:  $segid_num
atom_num[segid_num] $atom_num[$segid_num]\n";
    # Set the main record keeping variables
    $xyz[$segid_num]{$atom_num[$segid_num]} = "$res $num $atom
$x $y $z";

    # print "Segid: $segid  Segid_num: $segid_num  Atom_num
$atom_num[$segid_num]\n";
    # print "    For: $atom $res $num $x $y $z\n";

    # Total # of valid atoms in this segid:
    ++$atom_num[$segid_num];
}

# print "$atom_num[$segid_num]\n";

close IN;

# Check to make sure the last segment has SOMETHING in it!
if ($atom_num[$segid_num] == 0) {
    pop(@segid);
}
}

############################################################
# pdb_file_write
############################################################
sub pdb_file_write {
$total=@segid;
print "Total number of segments: $total\n";
if ($bogus_segid) {print "using bogus naming:\n";}
print @segid;
}
```

```
############################################################
# center_mass_calc
############################################################
sub center_mass_calc {
foreach $i (0 .. $atom_num[$segid_num]-1) {
    if ($use_mass eq "false") {$mass=1;}
    else{
    # Using exact atomic AMUs
    # Ideally we want to match 0 or more numbers first...
    $atom =~ s/^[0-9]*(\w)\w+/$1/;
    if ($atom =~ /C/) {$mass=12.011;}
    elsif ($atom =~ /H/) {$mass=1.00794;}
    elsif ($atom =~ /N/) {$mass=14.00674;}
    elsif ($atom =~ /O/) {$mass=15.9997;}
    elsif ($atom =~ /P/) {$mass=30.973762;}
    elsif ($atom =~ /S/) {$mass=32.066;}
    else {print "ERROR:\n";
        print "I don't know the mass of $atom\n";
    }
    }
    ($res, $num, $atom, $x, $y, $z) = split(/
/,$xyz[$segid_num]{$i});
    $total_x = $total_x + ($x*$mass);
    $total_y = $total_y + ($y*$mass);
    $total_z = $total_z + ($z*$mass);
    ++$total_num_atoms;
    $mass_total=$mass_total+$mass;
    # print "Total_x: $total_x Mass_total:$mass_total\n\n";
}
$center_x = $total_x/$mass_total;
$center_y = $total_y/$mass_total;
$center_z = $total_z/$mass_total;
}

############################################################
# inertia_matrix_prep
############################################################
sub inertia_matrix_prep {
foreach $i (0 .. $atom_num[$segid_num]-1) {
    ($res, $num, $atom, $x, $y, $z) = split(/
/,$xyz[$segid_num]{$i});
    if ($use_mass eq "false") {$mass=1;}
    else{
    # Using exact atomic AMUs
    # Ideally we want to match 0 or more numbers first...
    $atom =~ s/^[0-9]*(\w)\w+/$1/;
    if ($atom =~ /C/) {$mass=12.011;}
    elsif ($atom =~ /H/) {$mass=1.00794;}
    elsif ($atom =~ /N/) {$mass=14.00674;}
    elsif ($atom =~ /O/) {$mass=15.9997;}
    elsif ($atom =~ /P/) {$mass=30.973762;}
    elsif ($atom =~ /S/) {$mass=32.066;}
    else {print "ERROR:\n";
        print "I don't know the mass of $atom\n";
    }
    }

    # Transform the cartesian coordinates relative to the
center of mass.
    $x_prime=$x-$center_x; $y_prime=$y-$center_y; $z_prime=$z-
$center_z;

    # Build the inertia tensor 3x3 matrix, elements I11 I12
I13 I21 ...
    $r_sqrd = ($x_prime**2) + ($y_prime**2) + ($z_prime**2);
    $I11 = $I11 + $mass*($r_sqrd - ($x_prime**2));
    $I12 = $I12 + $mass*($x_prime*$y_prime);
    $I13 = $I13 + $mass*($x_prime*$z_prime);

    $I22 = $I22 + $mass*($r_sqrd - ($y_prime**2));
    $I23 = $I23 + $mass*($y_prime*$z_prime);

    $I33 = $I33 + $mass*($r_sqrd - ($z_prime**2));

    # Debugging area:
    # print "h: $i $x $y $z\n";
    # print "segid_num: $segid_num atom_num: $i\n";
    # print "X:$x Y:$y Z:$z Atom:$atom Mass:$mass\n";
}
}

############################################################
# inertia_calc
############################################################
sub inertia_calc {
$I21 = $I12;
$I31 = $I13;
$I32 = $I23;
```

```
# off diagnonal terms are negative:
$I12=$I12*(-1); $I13=$I13*(-1);
$I21=$I21*(-1); $I23=$I23*(-1);
$I31=$I31*(-1); $I32=$I32*(-1);

# print "Total X: $total_x  Total number of atoms:
$total_num_atoms\n";

open (OUT, "> temp_file");
print OUT "3
$I11 $I12 $I13
$I21 $I22 $I23
$I31 $I32 $I33\n";
close OUT;

@temp_result = `diag < temp_file`;
unlink (temp_file);

# remove carriage returns and blank spaces
foreach $h (1 .. 3) {$H[$h][$h] = $temp_result[$h-1];}
$count=0;
foreach $i (1 .. 3) {
    foreach $j (1 .. 3) {
    ++$count;
    $U[$i][$j] = $temp_result[$count+2];
    chop $U[$i][$j];
    $U[$i][$j] =~ s/^\s*([^\s]*)\s*$/$1/;
    }
}

}

############################################################
# vector_calc
############################################################

sub vector_calc {

# Which eigenvalue is smallest
if (($H[1][1] < $H[2][2]) && ($H[1][1] < $H[3][3])) {
    # print "The first eigenvalue (X) is the smallest!\n";
    # @norm=($H[1][1]/$H[1][1], $H[2][2]/$H[1][1],
$H[3][3]/$H[1][1]);
    @major = ($U[1][1], $U[2][1], $U[3][1]);
    @minor_1=($U[1][2], $U[2][2], $U[3][2]);
    @minor_2=($U[1][3], $U[2][3], $U[3][3]);
}
elsif (($H[2][2] < $H[1][1]) && ($H[2][2] < $H[3][3])) {
    # print "The second eigenvalue (Y) is the smallest!\n";
    # @norm=($H[1][1]/$H[2][2], $H[2][2]/$H[2][2],
$H[3][3]/$H[2][2]);
    @minor_1=($U[1][1], $U[2][1], $U[3][1]);
    @major = ($U[1][2], $U[2][2], $U[3][2]);
    @minor_2=($U[1][3], $U[2][3], $U[3][3]);
}
 elsif (($H[3][3] < $H[1][1]) && ($H[3][3] < $H[2][2])) {
    # print "The third eigenvalue (Z) is the smallest!\n";
    # @norm=($H[1][1]/$H[3][3], $H[2][2]/$H[3][3],
$H[3][3]/$H[3][3]);
    @minor_1=($U[1][1], $U[2][1], $U[3][1]);
    @minor_2=($U[1][2], $U[2][2], $U[3][2]);
    @major = ($U[1][3], $U[2][3], $U[3][3]);
}
# $center_x=0; $center_y=0; $center_z=0;
}

############################################################
# reset_inertia_prep
############################################################
sub reset_inertia_prep {
# reset totaling variables
$mass_total = 0;
$total_x = 0; $total_y = 0; $total_z = 0;
$center_x = 0; $center_y = 0; $center_z = 0;
$I11 = 0; $I12 = 0; $I13 = 0;
$I21 = 0; $I22 = 0; $I23 = 0;
$I31 = 0; $I32 = 0; $I33 = 0;
$total_num_atoms = 0;
}

############################################################
# report_short
# print out only the 3 principle axis components XYZ:
# This is handy for running from another script
############################################################
sub report_short {

print "$major[0] $major[1] $major[2]\n";
```

```
}

############################################################
# report
############################################################
sub report {

print
"\n============================================================\n
";
if ($analysis_type eq "all") {
    print "Calculation for pdbfile:$pdb_file segment: all\n";
}
else {
    print "Calculation for pdbfile: $pdb_file segment:
$segid[$segid_num]\n";
}
print "\nStarting matrix:\n";
$a="x"; $b="y"; $c="z"; write;
$a="---------"; $b="---------"; $c="---------"; write;
$a=$I11; $b=$I12; $c=$I13; write;
$a=$I21; $b=$I22; $c=$I23; write;
$a=$I31; $b=$I32; $c=$I33; write;

print "\nDiagonalized matrix (eigenvalues):\n";
$a="x"; $b="y"; $c="z"; write;
$a="---------"; $b="---------"; $c="---------"; write;
$a=$H[1][1]; $b=0; $c=0; write;
$a=0; $b=$H[2][2]; $c=0; write;
$a=0; $b=0; $c=$H[3][3]; write;

print "\nTransformation matrix (eigenvectors):\n";
$a="x"; $b="y"; $c="z"; write;
$a="---------"; $b="---------"; $c="---------"; write;
$a=$U[1][1]; $b=$U[1][2]; $c=$U[1][3]; write;
$a=$U[2][1]; $b=$U[2][2]; $c=$U[2][3]; write;
$a=$U[3][1]; $b=$U[3][2]; $c=$U[3][3]; write;

print "\nThe resultant MAJOR principle axis has:\n";
print "X components: $major[0]\n";
print "Y components: $major[1]\n";
print "Z components: $major[2]\n";
print "with:\n";
print "actual X:",$major[0]*50+$center_x,"\n";
print "actual Y:",$major[1]*50+$center_y,"\n";
```

```
print "actual Z:",$major[2]*50+$center_z,"\n";
print "and is centered at:\n";
print "X component: $center_x\n";
print "Y component: $center_y\n";
print "Z component: $center_z\n";
$f=@segid;
print "There are $f segments, segid_num is $segid_num\n";
print "with $atom_num[$segid_num] valid atoms in this
segment\n";
print
"============================================================\n";

}

############################################################
# write_vector_pdb_file
############################################################
sub write_vector_pdb_file {

# Write out a pdb file which represents the vector
open (PDB, ">$pdb_file.$ext");
# These values just fill the spaces:
$a=2; $b="  C"; $c="  C"; $d=1;
$x=$center_x; $y=$center_y; $z=$center_z;
write PDB;

$x=$major[0]*50+$center_x; $y=$major[1]*50+$center_y;
$z=$major[2]*50+$center_z;
write PDB;

$x=-$major[0]*50+$center_x; $y=-$major[1]*50+$center_y; $z=-
$major[2]*50+$center_z;
write PDB;

if ($xy eq "true") {
    $x=$minor_1[0]*10+$center_x; $y=$minor_1[1]*10+$center_y;
$z=$minor_1[2]*10+$center_z;
    write PDB;
    $x=$minor_2[0]*10+$center_x; $y=$minor_2[1]*10+$center_y;
$z=$minor_2[2]*10+$center_z;
    write PDB;
}

close PDB;
}
```

```
##########################################################
format STDOUT =
@>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
$a,$b,$c
.

format PDB =
ATOM  @>>>>>  @>>> @>>> @>>>>    @##.### @##.### @##.###
@<<<
$a, $b, $c, $d, $x, $y, $z, $segid
.
```