X-PLOR

(Version 4.0)

A System for X-ray Crystallography and NMR

Axel T. Brünger

The Howard Hughes Medical Institute and
Department of Molecular Biophysics and Biochemistry,
Yale University,
266 Whitney Avenue, P.O. Box 208114,
New Haven, CT 06520-8114

Contents

Acknowledgments

I am grateful for suggestions, comments, and contributions from many friends and colleagues. I also thank the users of X-PLOR for their valuable suggestions. Development of X-PLOR was supported in part by the Howard Hughes Medical Institute, the National Science Foundation, and the National Institutes of Health.

Bug Reports

You may send bug reports or comments to the author, preferably by e-mail:

```
xplor@laplace.biology.yale.edu
```

Distribution

X-PLOR is available directly from the author (Department of Molecular Biophysics and Biochemistry, Yale University, 266 Whitney Avenue, P.Ø. Box 208114, New Haven, CT 06520-8114, USA; phone: 203 432-6143; fax: 203 432-6946) for nonprofit ("academic") users. X-PLOR is currently used by several thousand researchers in over 1000 laboratories worldwide.

1 Prologue

X-PLOR is a program system for computational structural biology. X-PLOR stands for exploration of conformational space of macromolecules restrained to regions allowed by combinations of empirical energy functions and experimental data. But it also stands for exploration of modern concepts of structured programming in macromolecular simulation.

> As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem and now that we have gigantic computers, programming has become an equally gigantic problem. In this sense the electronic industry has not solved a single problem, it has only created them—it has created the problem of using its product. (E.W. Dijkstra, Turing Award Lecture, 1972)

X-PLOR's main focus is the three-dimensional structure determination of macromolecules using crystallographic diffraction or nuclear magnetic resonance (NMR) data. The program is based on an energy function approach: arbitrary combinations of empirical, geometric and effective energy terms describing experimental data may be used. The combined energy function can be minimized by a variety of gradient descent, simulated annealing, and conformational search procedures. The first version of X-PLOR (1.0) was published in the fall of 1987; it had evolved from a modified CRAY version of the CHARMM program (Brooks et al. 1983). X-PLOR was the first program to combine X-ray crystallographic diffraction data and molecular dynamics for refinement (Brünger, Kuriyan, and Karplus 1987). Since then the program has undergone extensive development, and the focus has shifted from refinement to structure determination. Major features of computational X-ray crystallography and solution NMR-spectroscopy have been included. Future development of X-PLOR is aimed at providing a comprehensive system for all computational aspects of macromolecular structure determination.

X-PLOR is more than a program: it is a macromolecular language. This flexible language allows the user to experiment with new ideas without being restricted to standard or "hard-wired" protocols. X-PLOR was designed to

provide user friendliness, machine portability, and highly efficient algorithms for modern computers.

The program has been implemented on nearly all modern computer types including vectorizing supercomputers (see Chapter 22). Furthermore, planning for a general parallel version of X-PLOR has begun. Since the program has been written in standard FORTRAN-77, it is usually straightforward to implement the program on a new machine. A suite of over 80 test cases, including input and output files, is provided as part of the distribution. These test cases are an integral part of X-PLOR that ensures continuing reliability and robustness of the system.

1.1 Electronic Location of Example Files

Most example input files listed in this book are located in subdirectories of the "tutorial" directory on the distribution medium. The fact that an example input file is available is indicated by the first line of the file, i.e.,

```
1 remarks  file xyz/junk.temp
```

where xyz refers to the subdirectory of the tutorial directory. The tutorial subdirectories also contain test data and output files obtained by running X-PLOR with these example input files.

1.2 Mathematica Interface

X-PLOR does not provide any direct graphical interfaces. Rather, it produces files that contain lists of numbers or other objects. Included in this book are many examples of how to interpret information from X-PLOR with Mathematica (Wolfram 1991) and to produce graphical illustrations. Mathematica version 2.0 or later is required. Mathematica can be obtained from Wolfram Research, Inc. (100 Trade Center Drive, Champaign, IL 61820, USA; phone: 217 398-0700).

1.3 Notation

Throughout this book, the Backus-Naur notation (Naur 1960) is used. Syntactic constructs are denoted by English words enclosed by angle brackets < and >; the possibility of a wildcard specification is indicated by enclosing the syntactic construct in <* *> angle brackets (Section 2.6). A definition of a new syntactic construct is indicated by the metasymbol ":==". Possible repetition of a syntactic construct is indicated by enclosing the construct within metabraces { and }. Please note that these braces ({,}) are also used by X-PLOR for comments. Optional (i.e., not always necessary) constructs are enclosed in square brackets [and]. Alternate constructs are separated

by the metasymbol |. Use of the | metasymbol is somewhat loose; it is left out where alternatives are specified on different printed lines, e.g.,

<dummy-statement>:== A | B | C

is equivalent to

<dummy-statement>:==

A text

B text

C text

Words or identifiers not enclosed in angle brackets should be typed as specified. Generally, uppercase letters are mandatory, whereas lowercase letters are optional. The equal sign "=" that is used in many assignments, e.g.,

```
1 set    message=on    end
```

is optional; i.e.,

```
1 set    message on    end
```

is a valid statement. However, in mathematical expressions (see Sections 2.14 and 2.16), the equal sign is mandatory.

Throughout this book, many actual example inputs for X-PLOR will be provided. To distinguish them from the explanatory text, they have been typeset in "typewriter" font with consecutive line numbering. The pointer

```
1 {===>}
```

indicates lines of the file that most likely need modification.

1.3.1 Example

Consider the following syntactic definition.

EXAMple <string> { <example-statement> } END

<example-statement>:==

CHARge=<real>
EXCLude=({ <atom> })
MASS=<real>
TESTing=A1 | A2 | A3
TYPE=<type>

<string>:== any sequence of four characters.

The following input lines are possibilities for valid constructs:

```
example A type=ABCD  mass=30.0 end
example B type=BBBB  charge=5.0  exclude=( A B C D ) end
example C charge=5. mass=40.0  type=XXX   testing=A3 end
example D end
```

1.4 Dimensions and Units

X-PLOR expects the following dimensions and units in all input files: Å for distances; kcal mole^{-1} for energy; psec for time, except for some specifications in the NMR relaxation statement, where it is in sec; amu (atomic mass units) for masses; K for temperature; and degrees for all angles, except for the specification of certain energy constants, which refer to kcal mole^{-1} rad^{-2}.

2 X-PLOR Language

X-PLOR has a powerful command parser that uses free-field input and structured control statements. It can be used both interactively and in batch or background mode. When X-PLOR is started, it expects to read the X-PLOR statements from standard input (in most cases FORTRAN unit 5).

2.1 Words

All X-PLOR statements are composed of words.

<**word**>:==

A one-character word.

A sequence of nonblank characters that contains no one-character words and that is enclosed by spaces or one-character words.

A quoted string, i.e., a sequence of any characters enclosed in double quotes (") (a double quote itself can be produced by """).

A one-character word comprises special letters:

<**one-character-word**>:==) | (| : | @ | =

In the case of mathematical expressions (such as vector statement and evaluate statement), one-character words constitute a larger set of special letters:

<**one-character-word**>:==) | (| : | @ | = | <
| > | # | * | ^ | ~ | + | - | /

Characters between comment braces { } or after an "!" on the same line are always ignored. The carriage return is treated as a space. Unless the word is a quoted string, all letters are converted to uppercase upon parsing.

2.1.1 Example

Suppose the following are two input lines for X-PLOR:

```
1 this is {comment}  a (test):   method=testing  procedure=
2 parsing   ! comment
```

The information that X-PLOR will receive from the parsing routines will then consist of a stream of words

```
1  THIS
2  IS
3  A
4  (
5  TEST
6  )
7  :
8  METHOD
9  =
10 TESTING
11 PROCEDURE
12 =
13 PARSING
```

In particular, the spacing and the carriage return have no influence on the interpretation of the input.

2.2 Numbers and Strings

The basic vocabulary of X-PLOR knows about numbers and character strings. Note that the following definitions are special cases of words.

<**complex**>:==(<**real**> , <**real**>) is a complex number.

<**integer**>:== is a positive or negative integer number, including zero.

<**logical**>:== is a logical variable, i.e., TRUE or FALSE. ON and OFF can be used in place of TRUE and FALSE, respectively.

<**real**>:== is a floating-point number.

<**string**>:== is a sequence of characters enclosed in double quotation marks. Note that in instances where the sequence of characters uniquely defines a word, the quotation marks can be left out. There are exceptions to this rule in the vector and evaluate statements; it is recommended to use quotation marks with strings in these statements.

2.3 Three-dimensional Vectors

The 3d-vector construct allows one to specify a three-dimensional real vector. The construct is also used to specify a 3×3 matrix by entering one row at a time (see Section 2.4).

<**3d-vector**>:==

(<**real**> , <**real**>, <**real**>) specifies the vector explicitly by using the x,y,z components. The commas are optional.

([TAIL=<selection>] [HEAD=<selection>]) defines the tail or the head of the vector as the center of mass of the specified atom selections using the mass atom property (Section 2.16) and the main coordinate set x,y,z (Section 6.1). If HEAD is not specified, it defaults to (0,0,0); similarly for TAIL.

2.4 3×3 Matrices

The matrix construct allows one to specify a 3×3 real matrix. There are several modes available to specify rotation matrices. An arbitrary 3×3 matrix can be specified by using the MATRix construct. Note: all rotations are counterclockwise when looking in the direction of the rotation axis.

<matrix>:==

AXIS <vector> <real> specifies input through the axis vector and the rotation angle κ.

EULEr <vector> specifies input through the Eulerian angles θ_1,θ_2,θ_3. θ_1 is the rotation around the z axis, θ_2 around the new x axis, and θ_3 around the new z axis. The maximum range is $0 < \theta_1 < 360°$, $0 < \theta_2 < 90°$, $0 < \theta_3 < 360°$.

LATTman <vector> specifies input through Lattman's angles ($\theta_+ = \theta_1 + \theta_3$, θ_2, $\theta_- = \theta_1 - \theta_3$). The maximum range is $0 < \theta_+ < 720°$, $0 < \theta_2 < 90°$, $0 < \theta_- < 360°$.

MATRix <vector> <vector> <vector> specifies direct input of the matrix by three 3d-vectors.

QUATernions <real> <real> <real> <real> specifies quaternions q_0, q_1, q_2, q_3, which are defined as

$$\begin{aligned} q_0 &= cos(\frac{\theta_2}{2})cos(\frac{\theta_1+\theta_3}{2}) \\ q_1 &= sin(\frac{\theta_2}{2})cos(\frac{\theta_1-\theta_3}{2}) \\ q_2 &= sin(\frac{\theta_2}{2})sin(\frac{\theta_1-\theta_3}{2}) \\ q_3 &= cos(\frac{\theta_2}{2})sin(\frac{\theta_1+\theta_3}{2}) \end{aligned} \tag{2.1}$$

with the constraint

$$q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1 \tag{2.2}$$

SPHErical <vector> specifies input through spherical polar angles ψ, ϕ, κ. ψ and ϕ specify the rotation axis. ψ is the inclination versus the y-axis; ϕ is the azimuthal angle, i.e., the angle between the x-axis and the projection of the axis into the x,z plane; and κ is the rotation around the rotation axis. The maximum range is $0 < \psi < 180°$, $0 < \phi < 180°$, $0 < \kappa < 360°$.

In the following example, a rotation matrix is specified by a rotation axis (2,3,4) and a rotation angle (40°) around the axis (counterclockwise rotation when looking in the direction of the rotation axis):

```
AXIS=( 2, 3, 4 )  40.
```

In the next example, a matrix is specified by direct input:

```
MATRix=( 1. 3. 5. )
       ( 4. 2. 1. )
       ( 2. 1. 8. )
```

The last example shows how to specify a rotation matrix by using the Eulerian angles θ_1,θ_2,θ_3:

```
EULEr=( 30. 40. 120. )
```

2.5 Symbols

A symbol is a word with a "$" as the first character. It is replaced by the actual value to which it has been assigned.

<symbol>:==

$<symbol-name> is a symbol with name <symbol-name>. Symbols can be assigned and manipulated by the evaluate statement (see Section 2.14) and several control statements (see Section 2.8). Symbols are sometimes declared during execution of particular applications. The data type can be real (floating point), string, or complex and is assigned automatically by the program. Integers are internally converted into floating point numbers. The <symbol-name> must contain fewer than 20 characters consisting of letters or numbers.

$<symbol-name> [**<FORTRAN-format>**] same as $<symbol-name> except that the specified FORTRAN format is used when the symbol is substituted. Example: $test[F10.4]. This will substitute the symbol using the FORTRAN format "F10.4". Only floating point formats are allowed. Symbols have to be of type real or complex.

Certain symbols are pre-defined upon startup of the program or are declared during execution of particular applications. Here are a few examples:

$? is not an assignable symbol; it produces a list of the currently assigned symbols.

$ANGL, $BOND, $DG, $DIHE, $ELEC, $ENER, $HARM, $IMPR, $NCS, $NOE, $PELE, $PLAN, $PVDW,$RELA, $VDW, $XREF, represent partial energy terms (see Sections 4.5 and 4.6). The data type is real. These symbols are declared upon evaluation of the X-PLOR energy function.

$CPU contains the CPU time (real).

$DATE contains the current date (string).

$EXIST_<symbol-name> is not an assignable symbol; it is a special symbol that returns the string "TRUE" if <symbol>=<symbol-name> has been declared previously; otherwise it returns "FALSE".

$GRAD specifies the rms value of the energy gradient. The data type is real. This symbol is declared upon execution of the X-PLOR energy function.

$KBOLTZ is declared during X-PLOR start-up. It contains the Boltzmann constant in units of the program (real).

$NAME is declared during X-PLOR start-up. It contains the username (string).

$PI is declared during X-PLOR start-up. It contains the value of π (real).

$RESULT contains the results after execution of various statements, such as "VECTor SHOW" and "COORdinates RMS END". Data type is either real or string, depending on the application.

$STATUS is declared during X-PLOR start-up. It provides status information about the read dynamics statement (string).

$SYSTEM is declared during X-PLOR start-up. It contains the system identification (string).

$TIME contains the wall-clock time (string).

2.5.1 Examples

A symbol $NEWSYMBOL is declared. Note that X-PLOR automatically determines the type of the symbol. This symbol is used in the subsequent statements. The third statement declares a symbol $GARBAGE. The fourth statement will display this symbol using FORTRAN format F8.2.

```
1  evaluate ($NEWSYMBOL=3.40+433^2)
2  xrefin wa=$NEWSYMBOL end
3  evaluate ($GARBAGE=sqrt($NEWSYMBOL))
4  display  $GARBAGE[F8.2]
```

In the next line, the symbol $NEWSYMBOL is redeclared as a character string.

```
1  evaluate ($NEWSYMBOL="testing 1 2 3")
```

As a consequence of this statement, $EXIST_NEWSYMBOL is set to "TRUE".

2.6 Wildcards

Another special case of a word is a wildcard.

<**wildcard**>:== { $*$ | % | # | + | <**string**> }

$*$ matches any string.

% matches a single character.

matches any string consisting of numerals.

+ matches a single character consisting of a numeral.

It is used to match a construct against sets of other words. The possibility of a wildcard specification is also indicated by enclosing syntactic constructs in <* *> angle brackets.

2.6.1 Example

The following atom selection selects all atoms that contain a "C" in their names at any position:

```
1  ( name *C* )
```

2.7 Filenames

A filename is any sequence of nonblank characters enclosed by spaces. In particular, the filename may contain one-character words, and it is case sensitive. This deviation from the general definitions of words and strings is done in order to allow for machine-specific filename specifications without having to enclose the filename in quotes.

2.7.1 Example

The following statements assign the specified files to the DISPlay unit:

```
1  SET DISPlay=USER1:[HOME.SUBDIRECTORY]TESTING_ABCDE.EXT;4 END
2  SET DISPlay=/mnt/usr/users/home/sub/testing.test END
```

On all UNIX and VAX/VMS systems, X-PLOR provides the feature of environmental variables or logical definitions. Suppose the user has defined a UNIX variable TOPPAR in his ".cshrc" file

```
1 setenv TOPPAR  /user/toppar
```

Then it is possible to access a file "xyz" in the directory "/user/toppar" in X-PLOR by specifying the filename "TOPPAR:xyz", e.g., if the user wants to set the DISPLay file,

```
SET DISPlay=TOPPAR:xyz
```

The same feature can be used on VAX/VMS systems by including a logical assignment statement in the user's login file:

```
$ ASSIGN/NOLOG user:[toppar] $TOPPAR
```

2.8 Control Statements

In general, a statement in X-PLOR can be a control statement or an application statement.

<X-PLOR-statement>:==

<control-statement>

<application-statement>

A control statement allows structured control of the sequence of application statements, such as loops, and conditional tests. It also allows switching the input stream to another file, opening and closing files, and various other operations.

<control-statement> :==

@<filename> deals with the fact that initially the parser reads from standard input (which will be FORTRAN unit 5 in most cases). The stream can be switched to another file by using this statement. Upon end of file, the parsing stream is switched back to the previous input. Nested streams are allowed.

@@<filename> has the same effect as the "@" statement, except when the stream file is invoked within a structured loop statement. In this case, the "@" statement inserts the contents of file filename into the loop and removes the statement in subsequent loop cycles, whereas the "@@" statement reads from filename each time the loop hits the statement. It should be noted that filename can be a symbol. This allows one to loop through a set of different filenames. Nested "@@" statements are allowed in this release of X-PLOR.

CLOSe <filename> DISPosition=KEEP|DELEte END explicitly closes a specified file. Normally this operation is done automatically by X-PLOR, so this statement should be used only in rare cases, such as closing and deleting a file.

DISPLAY <record> writes the record to a file that is specified by the "SET DISPlay" statement. The record can be any sequence of characters terminated by a carriage return. It may contain symbols that are substituted before the record is written to the file. If the last character

of the record is a backslash (
), the record can be continued on the next line.

EVALuate <evaluate-statement> manipulates symbols (see Section 2.14).

FOR <symbol> IN ({ <word> }) <basic-loop> assigns the symbol to a word from the specified set of words (one at a time) and executes the statements within the basic loop.

FOR <symbol> IN ID <selection> <basic-loop> assigns the symbol to the internal atom identifier for all atoms in the selection, and executes the statements within the basic loop (cf. Section 2.15). Care should be taken not to modify the molecular structure within the scope of the basic loop. The selection for the symbols is stored and computed when the loop is initialized, and it is not mapped when atom numbers change.

IF <condition> THEN <X-PLOR-statement>
[{ ELSEIF <condition> THEN <X-PLOR-statement> }]
[ELSE <X-PLOR-statement>] END IF depending on the conditions, sends control flow to the appropriate branch.

OPEN <filename> FORMatted = FORMATTED | UNFORMATTED ACCESS = READ | WRITE | APPEND END explicitly opens the specified file. Normally, this operation is done automatically by X-PLOR, and so the statement should be used only in rare cases, such as opening a file with append access.

REMARKS <record> writes the record to an internal title store. The record can be any sequence of characters terminated by a carriage return. It can contain symbols that are substituted before the record is stored. If the last character of the record is a backslash (
), the record can be continued on the next line. The internal title store is written to the first lines of output files.

REWInd <filename> END rewinds the specified file.

SET <set-statement> END sets various global parameters and options (see Section 2.13).

SYSTem <record> spawns a subprocess and executes the statement provided by the record. The record can be any sequence of characters terminated by a carriage return. It may contain symbols that are substituted before the record is written to the file. If the last character of the record is a backslash (
), the record can be continued on the next line. This statement is machine dependent. It may not be available on all operating systems.

WHILe <condition> <basic-loop> while the condition is true, executes the statements within the basic loop.

<condition> :== (**<word>** = |#| > | < |**GE**|**LE** **<word>**)
specify that a condition is true if the first word is equal to, not equal to, greater than, less than, greater than or equal to, or less than or equal to the second word, respectively.

<basic-loop> :== **LOOP <label>** { **<X-PLOR-statement>** [**EXIT <label>**] } **END LOOP <label>** represents a basic body of a loop. The label is a string with up to four characters. The EXIT statement allows jumping out of the specified loop (should be part of a conditional statement). Loops may be nested.

2.8.1 Example: A Conditional Test

This example either divides or multiplies the symbol $1 by a factor of two, depending on the value of the NOE energy ($NOE). In the first case, 40 steps of minimization are carried out, whereas in the latter case, 100 steps of minimization are carried out. Note that the indentation is arbitrary but can make the input file more readable.

```
if ($NOE > 10.0) then
   evaluate ($1=$1/2.0)
   minimize powell
     nstep=40
  end
else
   evaluate ($1=$1*2.0)
   minimize powell
     nstep=100
   end
end if
```

2.8.2 Example: A Simple Loop

The following example writes the characters a, b, c, d, and e to the file "testing.dat":

```
set display=testing.dat end
for $1 in ( a b c d e ) loop main
   display  $1
end loop main
```

2.8.3 Example: A Double Loop with Exit Condition

The conditional statement forces a user to exit both loops if the condition is satisfied.

```
for $1 in ( a b c d e ) loop m1
  while ($2 > 10.0 ) loop m2
     if ($3>1000.0) then  exit  m1  end if
     evaluate ($2=$2-1.0)
  end loop m2
end loop m1
```

2.8.4 Example: Switch Control to Another File

Suppose the file called "mini.str" contains the following statements:

```
minimize powell
   nstep=40.0
end
```

One can then include the contents of this file in another file by specifying the "@" statement; e.g.,

```
@mini.str
```

is equivalent to

```
minimize powell
   nstep=40.0
end
```

2.8.5 Example: Switch Control to Another File within Loops

When X-PLOR executes loops, it stores all input information in internal buffers. This may not be desirable if one wants to "loop through several files." To do this, one should use the "@@" statement. In the following example, four coordinate files are rms-compared to a set of reference coordinates:

```
coordinate disposition=comp @reference.pdb
for $1 in ( "coor1.pdb" "coor2.pdb" "coor3.pdb" "coor4.pdb" ) loop main
   coordinate @@$1
   coordinate rms end
end loop main
```

2.9 Application Statements

Application statements comprise all X-PLOR statements that generate, manipulate, and operate on the X-PLOR data structures. There are two types of application statements: simple application statements and structured application statements. In a simple application statement, the order of assignments is strict and has to be followed exactly as specified in the syntax. Examples of simple application statements are the vector statements and the evaluate statements. In the case of a structured application statement, the statement has to be terminated by an END keyword. The order in which assignments and statements are specified is not strict, although there may be special requirements for certain structured application statements. A structured application statement may also contain control statements. Examples are the dynamics Verlet statement, which executes molecular dynamics, and the xrefin statement, which sets up the data structures for crystallographic refinement. In this book it is always understood that structured application

statements may contain control statements, and explicit reference to control statements is omitted from the syntactic definitions. A list of all application statements is given at the end of the book.

2.10 Abbreviations

In most cases, keywords and qualifiers can be abbreviated to 4 characters. More characters can be used to make the input files more readable; e.g., instead of saying "XREF PRIN R END", one can say "XREFIN PRINT R END".

2.11 On-line HELP and Query

The X-PLOR statement and all structured application statements include the HELP and the "?" statement. The help statement provides information about the syntax and important remarks. The "?" statement provides information about the current status of parameters within the particular structured application statement. The "?" statements are not explicitly mentioned in the syntactic definitions.

2.12 Input and Output

For the sake of portability, most X-PLOR files are ASCII files. The record length of the ASCII is not more than 132 characters. One exception to this rule is the trajectory file (specified by the "TRAJectory=<filename> ISVFrq=<integer>" keywords, Section 10.1.6). This file is a binary file that can be converted to an ASCII file and vice versa by the dynamics merge statement. The use of ASCII files greatly simplifies communication between various types of computer equipment (workstations, minicomputers, supercomputers, etc.).

X-PLOR script files are free-field ASCII files with variable record length. The maximum record length is determined by the internal parameter COMMAX (Section 22.2), which defaults to 132 characters. The command file can be assigned to an interactive terminal. In this case, the user types the X-PLOR statements interactively.

2.13 Set Statement

The set statement allows one to change certain parameters that control X-PLOR program execution and output. The following is a list of the available statements:

<set-statement>:==

ABORt=OFF|NORMal|ALL determines whether program execution will be terminated in batch mode if an error is encountered. OFF means no termination in any case, NORMal means program terminates except in the case of minor errors, and ALL means program always terminates if an error is encountered. In interactive mode, i.e., if the error occurs due to a statement that was typed interactively, the error message will be printed without program termination (default: normal).

DISPlay-file=<filename> specifies an output file for DISPlay statement (Section 2.8). The display output can be redirected to the standard output by specifying "SET DISPlay=OUTPUT END" (default: OUTPUT).

ECHO=ON|OFF determines whether the input stream will be echoed to standard output. ON means input stream is echoed; OFF means echo is turned off. In interactive mode, the echo is always turned off.

INTEractive=ON|OFF deals with the fact that normally X-PLOR automatically determines whether the input comes from an interactive device. This flag overules the automatic assignment. Statements that are affected are ECHO and ABORt (default: depends on input device).

JOURnal=<filename> specifies a log-file for an interactive session. This statement allows one to specify a journal file during an interactive session that contains all statements that were typed (default: none).

MESSage=OFF|NORMal|ALL determines whether messages will be printed. OFF means only very important messages are printed. NORMal means most messages will be printed. ALL means all messages will be printed.

PRECision=<integer> specifies the number of significant digits for substitution of symbols representing real numbers to output; this does not affect the internal precision of the calculations (default: 6).

PRINt-file=<filename> specifies an output file for all PRINt statements (default: OUTPUT).

SEED=<real> is a seed for X-PLOR's internal random-number generator . It can be any positive real number.

TIMing=ON|OFF determines whether timing information is given for benchmarking X-PLOR. ON means timing information is given; OFF means no timing information is given (default: OFF).

2.14 Evaluate Statement

The evaluate statement allows one to carry out manipulations of symbols together with literal or numerical constants.

2.14.1 Syntax

EVALuate (<evaluate-statement>) is a control statement.

<evaluate-statement>:== **<symbol>** = **<operation>**

<operation>:== **<vflc>** [**<op>** **<operation>**]

<op>:==

+ denotes addition or concatenation for strings.

− denotes subtraction or unary minus or negative concatenation for strings.

∗ denotes multiplication.

/ denotes division.

ˆ denotes exponentiation.

∗∗ denotes exponentiation (same as ˆ).

<vflc>:== **<function>** | **<symbol>** | **<real>** | **<integer>** | **<string>** deals with the fact that the data types of the operations and operands have to match; otherwise an error message is issued. The available functions, such as SIN, COS, and TAN, are defined in Section 2.16.

2.14.2 Example

The statement

```
1 EVALuate ($1=1.0)
```

sets the symbol $1 to 1.0, whereas

```
1 EVALuate ($1=$1+2.2)
```

increases it by 2.2,

```
1 EVALuate ($1=$1*COS(2.*$PI$1))
```

sets it to 3.00498, and

```
1 EVALuate ($2="a"+"b"+encode($1))
```

sets the symbol $2 to the string “ab3.00498”.

2.15 Atom Selection

X-PLOR has a powerful atom selection syntax that allows one to select atoms without reference to the internal index and to construct arbitrary logical expressions of selected atoms. The number of selected atoms from the last executed selection statement is stored in the symbol $SELECT.

Atom selections are generally fragile, which means that information associated with atom selections is lost when changing the molecular structure (Sections 3.7, 3.8, 3.9, 3.10, and 3.11). This applies, for example, to the various selections in the xrefin statement (Section 12.3) and in the NOE statement (Section 18.1). However, certain selections are rendered only partially fragile by mapping the selected atoms to the new molecular structure after it has been modified. This applies to all atom properties (Section 2.16) except for the internal stores, which are fragile. It also applies to the atom-based parameter statements (Section 3.2.1). Note, however, that the atom selections are not applied to any newly created atoms and these atoms are then not selected. (The reason for this is that X-PLOR does not store the strings associated with the selections.)

2.15.1 Syntax

<selection>:== (**<selection-expression>**)

<selection-expression>:==

<term> selects atoms that belong to the term.

<term> { **OR** **<term>** } selects all atoms that belong to either one of the terms.

<term>:==

<factor> selects atoms that belong to the factor.

<factor> { **AND** **<factor>** } selects all atoms that belong to all of the factors.

<factor>:==

(**<selection-expression>**) selects all atoms that are selected in selection expression.

ALL selects all atoms.

<factor> AROUnd <real> selects all atoms that are within the specified real cutoff value around any selected atom in the factor.

ATOM <*segment-name*> <*residue-number*> <*atom*> selects all atoms that match the specified segment name, residue number, and atom name or wildcards of them. If the string is specified in double-quotes no wildcard matching is performed.

ATTRibute [ABS] <**property**> < | = | # | > <**real**> selects all atoms that have (absolute) properties less than, equal to, not equal to, or greater than the specified real number.

BYGRoup <**factor**> selects all atoms that belong to groups (see Section 3.1.1) containing at least one atom that has been selected in the factor.

BYRes <**factor**> selects all atoms that belong to residues containing at least one atom that has been selected in the factor.

CHEMical <***type***> selects all atoms that match the specified type (Section 3.1.1) or a wildcard of it. If the string is specified in double-quotes no wildcard matching is performed.

CHEMIcal <**type**>:<**type**> selects all atoms that have types greater than or equal to the first type but less than or equal to the second type. The order is defined by the ASCII collating sequence.

HYDRogen selects all atoms with masses (atom property MASS) approximately less than 3.5 amu.

ID <**integer**> selects all atoms that match the specified internal atom number. It should be used with caution. The main application is in conjunction with the "FOR <symbol> IN ID" statement (Section 2.8).

KNOWn selects all atoms with known coordinates.

NAME <***atom***> selects all atoms that match the specified atom name (Section 3.1.1) or a wildcard of it.

NAME <**atom**>:<**atom**> selects all atoms that have atom names greater than or equal to the first atom name but less than or equal to the second atom name. The order is defined by the ASCII collating sequence.

NOT <**factor**> selects all atoms that have not been selected in the factor.

POINt <**3d-vector**> **CUT** <**real**> selects all atoms that are within the specified real cutoff value around the specified 3d-vector (Section 2.3).

PREVious selects all atoms that have been selected in a previous selection in application statements that contain multiple selections.

RESIdue <***residue-number***> selects all atoms that match the specified residue number (Section 3.7) or a wildcard of it. If the string is specified in double-quotes no wildcard matching is performed.

RESIDue <**residue-number**>:<**residue-number**> selects all atoms that have residue numbers greater than or equal to the first residue number but less than or equal to the second residue number. The order is defined by the numerical sequence of the residue numbers. If a pair of residue numbers is identical apart from different insertion characters

(e.g., 15A and 15B), the order is defined by the ASCII collating sequence of the insertion characters (i.e., 15A less than 15B). Please note that the ASCII collating sequence is case sensitive. Contrary to intuition this implies that "a" is greater than "A".

RESName <*residue-name*> selects all atoms that match the specified residue name (Section 3.1.1) or a wildcard of it. If the string is specified in double-quotes no wildcard matching is performed.

RESName <residue-name>:<residue-name> selects all atoms that have residue names greater than or equal to the first residue name but less than or equal to the second residue name. The order is defined by the ASCII collating sequence.

<factor> SAROund <real> selects all atoms that are within the specified real cutoff value around any selected atom in the factor or any of its crystallographic or non-crystallographic symmetry mates. (See Section 12.3 for the definition of crystallographic symmetry and Chapter 16 for the definition of non-crystallographic symmetry.)

SEGIdentifier <*segment-name*> selects all atoms that match the specified segment name (Section 3.7) or a wildcard of it. If the string is specified in double-quotes no wildcard matching is performed.

SEGIdentifier <segment-name>:<segment-name> selects all atoms that have segment names greater than or equal to the first segment name but less than or equal to the second segment name.

STORE1 | STORE2 | STORE3 | STORE4 | STORE5 | STORE6 | STORE7 | STORE8 | STORE9 selects all atoms for which the value of STOREi is greater than 0; e.g., STORE2 is short hand for "ATTRibute STORE2 > 0", etc. The STOREi can be defined by the vector ID statement or the vector statement (see Section 2.16).

TAG selects exactly one atom from each residue. These selected atoms may be used to "tag" all residues without having to refer to residue numbers or identifiers. The sequence of selected atoms is determined by the order in which the residues have been created through the segment statement (see Section 3.7).

<property>:== B | BCOMp | CHARge | DX | DY | DZ | FBETa | HARM | MASS | Q | QCOMp | REFX | REFY | REFZ | RMSD | VX | VY | VZ | X | XCOMp | Y | YCOMp | Z | ZCOMp is a group of properties defined in Section 2.16.

2.15.2 Examples

The first example selects all C^{α} carbon atoms between residue number 40 and 100:

```
1 ( name ca and resid 40:100 )
```

The next example selects all heavy side-chain atoms:

```
( not ( name ca or name n or name c or name o or hydrogen ) )
```

The next example selects all atoms in Phe residues that are within 20 Å around residue 1:

```
( resname phe and ( residue 1 around 20.0 ) )
```

The next example is similar to the previous one, except that all atoms of a particular Phe residue are selected once any atom of this residue is within 20 Å from residue 1:

```
( byresidue ( resname phe and ( residue 1 around 20.0 ) ) )
```

Suppose that we want to get the stereochemistry as a function of residue number. The following example tags each residue by using its C^{α} carbon atom and then computes the rms deviation of bond lengths from ideality for all atoms of the selected residue.

```
for $1 in id ( name ca ) loop main
   constraints interaction=( byresidue ( id $1 ) )=(previous) end
   print threshold=0.1 bonds
   display   $RESULT
end loop main
```

2.16 Vector Statement

The vector statement allows one to manipulate atomic properties, such as masses, charges, coordinates, forces, and atom names. Mathematical expressions can be constructed that involve atomic properties. The vector statement can also be used to analyze atomic properties and to transfer the information to the $RESULT symbol. Finally, the vector statement can be used to define and store an atom selection that can be recalled later.

2.16.1 Syntax

VECTor <**vector-statement**> is invoked from the main level of X-PLOR.

<**vector-statement**>:==

<**vector-mode**> <**vector-expression**> <**selection**>

<**vector-mode**>:==

DO manipulates atom properties. The operations are carried out component by component for all selected atoms.

IDENtify is used to define and store an atom selection. The output array will contain the sequential number of the selected atoms or otherwise zero.

SHOW <**vector-show-property**> can be used to analyze atom properties.

<**vector-show-property**>:==

AVE shows the arithmetic average of selected elements and stores it in $RESULT.

ELEMent shows selected elements and stores the last element in $RESULT.

MAX shows the maximum of selected elements and stores it in $RESULT.

MIN shows the minimum of selected elements and stores it in $RESULT.

NORM shows the norm ($\sqrt{\sum x_i^2/N}$) of selected elements and stores it in $RESULT.

RMS shows the root-mean-square (rms) deviation

$$\sqrt{(\sum (x_i - < x >)^2)/N} \tag{2.3}$$

of selected elements and stores it in $RESULT.

SUM shows the arithmetic sum of selected elements and stores it in $RESULT.

<**vector-expression**>:==

<**atom-property**> = <**vector-operation**> carries out the vector operation and assigns it to the specified atom property.

<**vector-operation**> carries out the vector operation without assigning the result to an atom property. It should be used for the IDENtify and SHOW vector modes.

<**vector-operation**>:==

<**vflc**> [<**op**> <**vector-operation**>] executes operators with the highest precedence first. Operators with the same precedence are executed from left to right. Operations have to be meaningful; i.e., the data type of the operands has to match the operation. For strings, only the "+" and "-" operations are allowed. The following is a list of the operators with increasing precedence:

<**op**>:==

\+ denotes addition; concatenation for strings.

– denotes subtraction; unary minus or negative concatenation for strings.

∗ denotes multiplication.

/ denotes division.

^ denotes exponentiation.

∗∗ denotes exponentiation (same as ^).

<vflc>:==

<atom-property> | **<function>** | **<integer>** | **<real>** | **<string>** | **<symbol>** is a group of functions. Use of a string requires enclosure in double quotes“ ”. The data type of the function arguments has to match the data type of the operands.

<function>:==

ABS(<vflc>) expects one argument and returns its absolute value. Argument restrictions: no string.

ACOS(<vflc>) denotes arc cosine. Argument restrictions: no string or complex.

ASIN(<vflc>) denotes arc sine. Argument restrictions: no string or complex.

COS(<vflc>) denotes cosine. Argument restrictions: no string; expects argument in degrees.

DECODE(<vflc>) converts a character string to a numerical number if possible.

ENCODE(<vflc>) converts a numerical number to a character string.

EXP(<vflc>) is an exponentiation function. Argument restrictions: no string.

GAUSS(<vflc>) is a Gaussian distribution random-number function; it has one argument, the desired standard deviation. The mean of the distribution is always zero. Argument restrictions: no string or complex.

IMOD(<vflc>,<vflc>) returns the remainder of the first argument divided by the second after applying the NINT operator to the arguments. Argument restrictions: no string or complex.

INT(<vflc>) is a truncation. Argument restrictions: no string.

LOG10(<vflc>) is a natural log. Argument restrictions: no string or complex; argument must be greater than zero.

LOG(<vflc>) is a logarithmic function. Argument restrictions: no string or complex; real numbers must be greater than zero, and the complex (0.0,0.0) is illegal.

MAX(<vflc> {, <vflc> }) is a maximum-value function; it must have at least two arguments, and it returns the value of the argument with the maximum value. Argument restrictions: no string or complex.

MAXW(<vflc>) is a Maxwellian distribution random-number function; it has one argument, the desired standard deviation. The mean of the distribution is always zero. Argument restrictions: no string or complex.

MIN(<vflc> {, <vflc> }) is a minimum-value function; it must have at least two arguments, and it returns the value of the argument with the minimum value. Argument restrictions: no string or complex.

MOD(<vflc>,<vflc>) returns the remainder of the first argument divided by the second. Argument restrictions: no string or complex.

NORM(<vflc>) is a normalization function; it has one argument, which must be a recognized variable. This function calculates the sum of the squares of all selected elements in the argument array. Then it divides each selected element by the square root of the sum of the squares. Argument restrictions: no string or complex.

NINT(<vflc>) is the nearest-integer truncation. Argument restrictions: no string.

RANDom() is a random-number function; it has no argument. It returns a uniform distribution between 0 and 1.

SIGN(<vflc>) is a transfer of sign. If the argument is $>= 0$, it returns $+1$; if the argument is < 0, it returns -1. Argument restrictions: no string or complex.

SIN(<vflc>) denotes sine. Argument restrictions: no string; expects degrees.

SQRT(<vflc>) returns the square root of the given argument. Argument restrictions: no string; no negative real numbers.

STEP(<vflc>) is a step function; it expects one real-number argument. If the argument is greater than zero, heavy returns a one; otherwise heavy returns a zero. Argument restrictions: no string or complex.

TAN(<vflc>) denotes tangent in degrees. Argument restrictions: no string or complex.

<atom-property>:==

B B-factors of main coordinate set in $Å^2$ (real)

BCOMp B-factors of comparison coordinate set in $Å^2$ (real)

CHARge electric charge in electronic charges (real)

CHEMical chemical atom type (string)

DX x component of first derivatives in kcal $mole^{-1}$ $Å^{-1}$ (real)

DY y component of first derivatives in kcal mole^{-1} Å^{-1} (real)

DZ z component of first derivatives in kcal mole^{-1} Å^{-1} (real)

FBETa friction coefficient in psec^{-1} (real)

HARMonic energy constants of harmonic restraints in kcal mole^{-1} Å^{-2} (real)

MASS mass in amu (real)

NAME atom name (string)

Q occupancies of main coordinate set (real)

QCOMp occupancies of comparison coordinate set (real)

REFX x component of reference coordinate set in Å (real)

REFY y component of reference coordinate set in Å (real)

REFZ z component of reference coordinate set in Å (real)

RESId residue number (string)

RESName residue name (string)

RMSD array used by various modules, e.g., the COOR RMS statement

SEGId segment or chain identifier (string)

STORE1 1st internal store, is fragile (real)

STORE2 2nd internal store, is fragile (real)

STORE3 3rd internal store, is fragile (real)

STORE4 4th internal store, is fragile (real)

STORE5 5th internal store, is fragile (real)

STORE6 6th internal store, is fragile (real)

STORE7 7th internal store, is fragile (real)

STORE8 8th internal store, is fragile (real)

STORE9 9th internal store, is fragile (real)

VX x component of current velocities in Å psec^{-1} (real)

VY y component of current velocities in Å psec^{-1} (real)

VZ z component of current velocities in Å psec^{-1} (real)

X x component of main coordinate set in Å (real)

XCOMp x component of comparison coordinate set in Å (real)

Y y component of main coordinate set in Å (real)

YCOMp y component of comparison coordinate set in Å (real)

Z z component of main coordinate set in Å (real)

ZCOMp z component of comparison coordinate set in Å (real)

2.16.2 Requirements

The molecular structure has to be present. Upon modification of the molecular structure (e.g., delete or patch statement), the contents of the internal stores are destroyed; i.e., they are fragile (Section 2.15). However, all other atom properties are conserved; i.e., they are only partially fragile (Section 2.15).

2.16.3 Examples

The first example divides the coordinate array Z by the derivative array DX, adds the quotient to the coordinate array Y, and stores the result in the coordinate array X. The operations are carried out component by component for all atoms.

```
1 vector do ( X = Y + Z / DX ) ( all )
```

The next example computes a Gaussian distribution with standard deviation 1.0 and stores the result in the coordinate array x for all C^{α} atoms:

```
1 vector do ( X = GAUSS( 1.0 ) ) ( name ca )
```

The next example provides a listing of the X coordinates of all Tyr residues:

```
1 vector show element ( X ) ( resname tyr )
```

The next example computes the average of all electric charges in residue 34. This average value is then stored in the symbol $1 by using the evaluate statement.

```
1 vector show ave ( charge ) ( residue 34 )
2 evaluate ($1=$RESULT)
```

The next example stores the specified atom selection in the array STORE1:

```
1 vector identity ( store1 ) ( attribute mass > 30.0 )
```

The array STORE1 will be nonzero for the selected atoms and zero otherwise. The values for the selected atoms represent the sequential number of the selected atoms. The array STORE1 can be recalled by using

```
1  ( store1 )
```

in a selection statement.

3 Topology, Parameters and Molecular Structure

3.1 Topology Statement

The topology statement consists of information about "residues" that combine to form a particular macromolecule. The data include descriptions of atoms, assignments of covalent bonds (connectivity), bond angles, and other information. The information provided by the topology statement is used by the segment statement and the patch statement to generate the molecular structure.

X-PLOR, like many other macromolecular mechanics/dynamics packages, can use a classification scheme that employs chemical atom types. This system of atom classification makes force fields transferable between groups that are almost equivalent and thereby allows one definition to work for all or almost all instances of that type of atom. The specific atom classifications can be arbitrarily defined by the user. The parameters of chemical atom types are given in the parameter statement (Section 3.2.1). Electric charges are specified on an individual atom basis, since they are less transferable than stereochemical parameters. For historical reasons, atomic masses may also be specified on an individual atom basis, but there exists the mass statement outside the residue statement that assigns default atomic masses on a basis of chemical atom type. Apart from the type-based parameter retrieval, X-PLOR also provides atom-based parameter statements (Section 3.2), which allow one to specify any parameter on a per-atom basis.

3.1.1 Syntax

TOPOlogy { <**topology-statement**> } **END** is invoked from the main level of X-PLOR.

<**topology-statement**>:==

AUTOgenerate ANGLe=<**logical**> **DIHEdral=**<**logical**> **END** works only on regular residues, not on patched ones. It automatically generates all possible bond angles based on the connectivity list of the

particular residue. It operates on the residues following the statement until the next autogenerate statement appears.

MASS=<type>=<real> adds a default mass assignment for the specified type of atom to the topology database. The real number is defined in atomic mass units. This value will be used by default for all atoms of that type unless an explicit MASS statement is given in the atom statement (see below).

RESEt erases all previous entries into the topology database.

RESIdue <residue-name> { **<residue-statement>** } **END** adds a residue to the topology database.

PRESidue <residue-name> { **[ADD | DELEte | MODIfy] <residue-statement>** } **END** adds a patch residue to the topology database.

<residue-statement>:==

ACCEptor <atom> <atom> adds a hydrogen-bond acceptor for the explicit hydrogen-bonding energy term or for purposes of analysis to the residue database. The first atom is acceptor atom; the second atom is antecedent atom and can be the string NONE.

ANGLe <atom> <atom> <atom> adds a bond angle made by the three atoms to the residue database. It should not be used if autogenerate angles are active.

ATOM [<patch-character>] <atom> <atom-statement> END adds an atom to the residue database. The statement defines atom as a four-character string specifying the name, and atom statement specifies atom type, charge, and mass. The patch character is a 1-character string and may be used only for PRESidue.

BOND <atom> <atom> adds a covalent bond between the specified atoms to the residue database. The atoms have to be defined previously by atom statements. Parameters for the bond are specified in Section 3.2.1; Eq. 4.4 describes the bond energy.

DIHEdral <atom> <atom> <atom> <atom> [MULTiple <integer>] adds a dihedral angle to the residue database. The dihedral angle (ijkl) is defined by the angle between the plane made by the atoms (ijk) and the plane made by the atoms (jkl). The dihedral statement should not be used if autogenerate dihedrals are active, unless multiple dihedral angles are wanted. MULTiple specifies m dihedral angle entries for the same instance of four atoms (Eq. 4.6). This allows one to specify a cosine expansion for the dihedral angle potential energy. It must be accompanied by a corresponding DIHEdral angle parameter entry with appropriate multiplicity. Parameters for the dihedral angle are specified in Section 3.2.1; Eq. 4.6 describes the dihedral energy.

DONOr <atom> <atom> adds a hydrogen-bond donor for the explicit hydrogen-bonding energy term or for purposes of analysis to the residue database. The first atom is hydrogen atom, and the second atom is heavy atom; the first atom may be the string NONE for extended atom force fields. Parameters for the explicit hydrogen bonds are specified in Section 3.2.1; Eq. 4.25 describes the hydrogen-bond energy.

GROUP partitions the atoms into nonbonded groups (see Section 4.3.3).

IMPRoper <atom> <atom> <atom> <atom> [MULTiple <integer>] adds an improper angle to the residue database. The definition is identical to the DIHEdral angle, but it uses a different set of parameters (cf. Section 3.2.1). As in the case of the DIHEdral angle, the improper angle (ijkl) is defined by the angle between the plane made by the atoms (ijk) and the plane made by the atoms (jkl). MULTiple specifies m improper angle entries for the same instance of four atoms (Eq. 4.6). This allows one to specify a cosine expansion for the improper angle potential energy. It must be accompanied by a corresponding IMPRoper angle parameter entry with appropriate multiplicity. Parameters for the improper angle are specified in Section 3.2.1; Eq. 4.6 describes the improper energy.

<atom>:== is the name of the atom; it is a four-character string.

<atom-statement>:==

CHARge=<real> specifies an electrostatic charge in units of one electron charge. This affects the electrostatic energy (Eq. 4.16).

EXCLude=({ <atom> }) specifies explicit nonbonded interaction exclusions (see Section 4.3.3). The atoms have to have been defined previously by an atom statement.

MASS=<real> overwrites a default given by mass statement outside residue statement, in atomic mass units.

TYPE=<type> specifies the chemical atom type, a string with up to four characters. The atom type is used to retrieve type-based parameters (see Section 3.2.1).

<type>:== is any sequence of four characters.

3.1.2 Example: Topology of a Leucine Amino Acid

The following example shows how to set up a Leu amino acid:

```
1 TOPOlogy
2    MASS   H      1.008
3    MASS   C     12.011
4    MASS   CH1E  13.019
5    MASS   CH2E  14.027
6    MASS   CH3E  15.035
```

```
7     MASS   N     14.0067
8     MASS   O     15.9994
9
10    AUTOgenerate ANGLes=TRUE END
11
12    RESIdue LEU
13      GROUp
14      ATOM N    TYPE=NH1   CHARge=-0.35   END
15      ATOM H    TYPE=H     CHARge= 0.25   END
16      ATOM CA   TYPE=CH1E  CHARge= 0.10   END
17      ATOM CB   TYPE=CH2E  CHARge= 0.00   END
18      ATOM CG   TYPE=CH1E  CHARge= 0.00   END
19      ATOM CD1  TYPE=CH3E  CHARge= 0.00   END
20      ATOM CD2  TYPE=CH3E  CHARge= 0.00   END
21      ATOM C    TYPE=C     CHARge= 0.55   END
22      ATOM O    TYPE=O     CHARge=-0.55   END
23
24      BOND N    CA
25      BOND CA   C
26      BOND C    O
27      BOND N    H
28      BOND CA   CB
29      BOND CB   CG
30      BOND CG   CD1
31      BOND CG   CD2
32
33      DIHEdral N    CA   CB   CG
34      DIHEdral CA   CB   CG   CD2
35
36      IMPRoper CA   N    C    CB
37      IMPRoper CG   CD2  CD1  CB
38
39      DONOr H    N
40      ACCEptor O C
41    END
42  END
```

3.2 Parameter Statement

The parameter statement specifies various constants for the conformational and nonbonded energy terms (Section 4.1) and for a number of other X-PLOR applications, such as distance geometry (Section 19).

Figure 3.1 illustrates the data flow for the parameters. X-PLOR stores and manipulates the parameter information in two forms: "type-based" and "atom-based". A type-based parameter is characterized by the chemical types of the atoms involved; an atom-based parameter is characterized by the individual atoms involved. The chemical types of the atoms are specified in the topology statement (Section 3.1.1) and can be manipulated with the vector statement (Section 2.16). The chemical types usually map chemically equivalent atoms to the same type; e.g., all carbonyl oxygen atoms are mapped to the type "O". This implies that there are usually many more atoms in the molecular structure than there are atom types. The use of atom types can thus reduce the number of parameters that need to be specified for a molecular structure. However, this convenience could pose a problem when the chemical environment of a particular atom induces distortions of the expected "ideal" geometry for the corresponding atom type. It would

not necessarily be a good idea to introduce a new atom type, as the number of atom types is limited. To get around this problem, X-PLOR supports atom-based parameter information. This implies that, at least in principle, one could have a different parameter for each interaction between particular atoms. Atom-based parameters are possible for bonds, bond angles, dihedral angles, improper angles, and van der Waals interactions. They are presently not supported for the explicit hydrogen-bonding energy term.

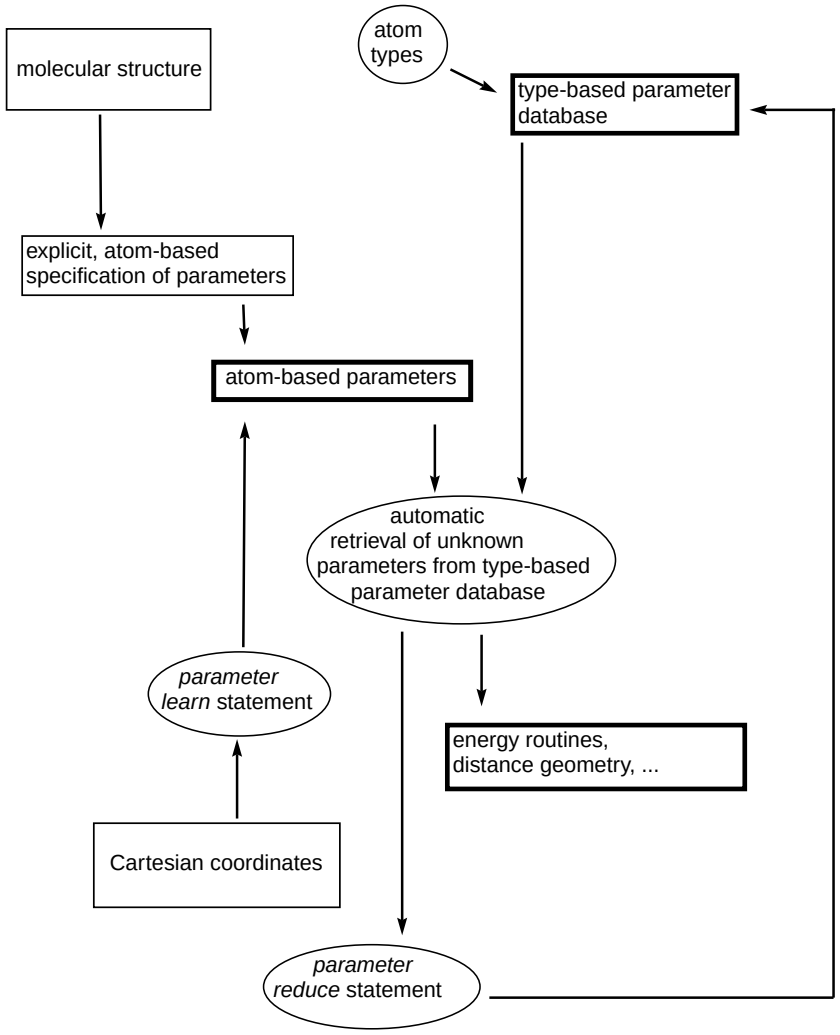

Figure 3.1: Flow of parameters.

X-PLOR allows one to mix atom-based and type-based parameters. Atom-based parameters always take precedence over type-based parameters. The atom-based parameter information can be obtained through appropriate ex-

plicit parameter statements, such as

```
bond ( name c* ) ( name n* ) 500. TOKEN
```

or through the Cartesian coordinates in the main coordinate set in combination with the learn statement. This atom-based information is augmented with the type-based information each time an X-PLOR application requests it; e.g., the call to an energy routine triggers an operation that retrieves all parameters not set by atom-based information from the type-based parameter database. In this way either atom-based parameters, type-based parameters, or a combination of both can be used. Atom-based parameters can be changed or added at any time during program execution. Type-based parameters cannot be manipulated, but additional entries can be added or the database erased and reinitialized. The mapping of atoms to chemical types can be manipulated by applying the vector statement to the chemical atom property (Section 2.16). All these changes will take effect the next time the particular parameter is requested by an X-PLOR application.

The learn statement can be used to generate atom-based parameters for interactions involving selected atoms. Parameters can be learned from a single coordinate set or from an ensemble of coordinates. Equilibrium constants for bonds, bond angles, dihedral angles, or improper angles can be learned. Energy constants can be obtained from the standard deviation of the equilibrium parameters around their mean. Clearly, this is only possible when an ensemble average of coordinate sets is available. Improper and dihedral angles are learned under the assumption that the periodicity n is zero (Eq. 4.6) regardless of any typed-based default periodicity. It should be noted that the generation of a template structure (Section 20.1) is essentially the inverse of the learn statement: it creates a molecular conformation with ideal geometry from the parameters.

The reduce statement tries to map the atom-based parameters into type-based parameter information using the chemical atom property. The redundancy of this process implies that the parameter information needs to be averaged. Existing type-based parameters can be overwritten or stay untouched, depending on the specified option. The newly created type-based parameter set can be written to a parameter file for use with other X-PLOR protocols. Only bond, bond angle, dihedral angle, and improper angle parameters can be learned or reduced. Energy constants can be obtained either by averaging the atom-based values or by calculating the standard deviation of the equilibrium values from their mean for a particular instance of type-based parameters.

The parameter specifications are insensitive to the order in which the atoms are specified; e.g.,

```
bond a b 10.0 1.0
```

and

```
1  bond b a 10.0 1.0
```

are seen as identical parameter entries by X-PLOR.

The constants of the parameter statement are dimensioned such that Å is the unit of length, kcal mole^{-1} is the unit of energy, the atomic mass unit is the unit of mass, and the charge of one electron is the unit of electric charge.

Obtaining empirical parameters for macromolecules is a difficult task. To describe the development of parameters is beyond the scope of this book. X-PLOR includes a number of parameter files for proteins, nucleic acids, and other biological macromolecules (Section 3.6). Fortunately, structure determination and refinement of crystallographic or NMR-derived structures are usually not very sensitive to the exact choice of energy constants. The learn statement provides a convenient tool to obtain the equilibrium geometry for chemical compounds that are not included in the parameter files. The equilibrium parameters can be obtained from known Cartesian coordinates of the compounds. The energy constants can be set to uniform values or can be derived from a statistical analysis of the structures of several related compounds. Note that the learn statement does not actually create new interaction terms (bonds, bond angles, improper angles, dihedral angles) for the molecular structure. It simply assigns parameters for interaction terms that are already defined in the molecular structure. The creation of new interaction terms is described in Section 3.8.

3.2.1 Syntax

PARAmeter {<**parameter-statement**> } **END** is invoked from the main level of X-PLOR.

<**parameter-statement**>:==

ANGLe <**type**> <**type**> <**type**> <**real**> <**real**> [**UB** <**real**> <**real**>] adds a bond angle parameter set for the three atom types to the parameter database. The first real specifies k_ϕ, which has units of kcal mole^{-1} rad^{-2}, and the second real specifies θ_0, the equilibrium angle, which has units of degrees (Eq. 4.5). The optional UB specification activates the Urey-Bradley term (Eq. 4.5), where the first real is the Urey-Bradley energy constant k_{ub} and the second real is the Urey-Bradley equilibrium distance r_{ub} between the first and the third atom that define the angle. If UB is not specified, the Urey-Bradley equilibrium distance and energy constant default to zero. The program automatically performs an interchange of the first with the third atom types where this is required.

ANGLe <**selection**> <**selection**> <**selection**> <**real**> <**real**> [**UB** <**real**> <**real**>] is an atom-based version of the ANGLe statement

that specifies the bond angle parameters to be used for any angles in the molecular structure that match the given triple atom selection. It can apply to more than one angle, depending on the number of angles that match the triple atom selection. The definition of the reals is identical to the type-based angle statement. The energy and equilibrium constants can be set to the string TOKEn, which implies that the corresponding parameter will be untouched by this statement.

BOND <**type**> <**type**> <**real**> <**real**> adds a covalent bond parameter set for the two atom types to the parameter database. The first real specifies k_b, which is the energy constant in units of kcal mole^{-1} Å^{-2}, and the second real specifies r_0, which is the equilibrium bond length in Å (Eq. 4.4). The program automatically performs an interchange of the two atom types where this is required.

BOND <**selection**> <**selection**> <**real**> <**real**> is an atom-based version of the BOND statement. The definition of the reals is identical to the type-based bond statement. It can apply to more than one bond, depending on the number of bonds that match the double atom selection. The energy and equilibrium constants can be set to the string TOKEn, which implies that the corresponding parameter will be untouched by this statement.

DIHEdral <**type**> <**type**> <**type**> <**type**> [**MULT** <**integer**>] {<**real**> <**integer**> <**real**> } adds a dihedral angle parameter set for the four atom types to the parameter database (see also Eq. 4.6). The MULT option specifies the multiplicity m of the dihedral angle (default: m=1). For multiple dihedrals of multiplicity m, there are m groups of 3 items following the MULT <integer> statement. The first real of each group specifies k_θ, the integer is the periodicity n, and the second real specifies δ, the phase-shift angle, which has units of degrees. If the periodicity n is greater than 0, k has the units of kcal mole^{-1}; if the periodicity is 0, k has the units of kcal mol^{-1} rad^{-2} (Eq. 4.6). The special character **X** is reserved for the following combination: X <type> <type> X; it acts as a wildcard. The parameter retrieval for a specified dihedral angle proceeds in the following way: first, a match without wildcards is attempted, and second, a match against dihedral parameters containing wildcards is attempted. Some instances (such as sugars) require the mixing of multiple dihedral angle terms with different periodicities. In this case, dihedral statements with the multiple option should be given in the parameter statement and in the topology statement (Section 3.1.1). Wildcards are not allowed for multiple dihedral angles. The program automatically performs the interchange (a b c d) $\rightarrow$ (d c b a) where this is required.

DIHEdral <**selection**> <**selection**> <**selection**> <**selection**> [**MULT** <**integer**>] { <**real**> <**integer**> <**real**> } is an atom-based version of the DIHEdral statement. The definition of the real

and integer numbers is identical to the DIHEdral definition (see also Eq. 4.6). For multiple dihedrals of multiplicity m, there are $3m$ items following the MULT <integer> statement (i.e., m sets of one energy constant, one periodicity, and one offset). The statement can apply to more than one dihedral angle, depending on the number of bonds that match the quadruple atom selection. An atom-based statement takes priority over any type-based parameter that may have been specified earlier. The energy constants, periodicities, and offsets can be set to the string TOKEn, which implies that the corresponding constant will be untouched by this statement.

HBONded <***type***> <***type***> <**real**> <**real**> adds an explicit hydrogen-bonding parameter set for the specified pair of atom types to the parameter database. The first atom type refers to donor (heavy) atoms, whereas the second one refers to the acceptor atoms. Wildcards are allowed for both donor and acceptor atom types. The first real is the well depth $\frac{-B^2}{4A}$ and the second is the distance $\sqrt[6]{\frac{2A}{B}}$ (Eq. 4.25).

HBONDS { <**hbonds-statement**> } **END** specifies global hydrogen-bond parameters for the explicit hydrogen-bond energy term (see Eq. 4.25).

IMPRoper <**type**> <**type**> <**type**> <**type**> [**MULT** <**integer**>] {<**real**> <**integer**> <**real**> } adds an improper angle parameter set for the four atom types to the parameter database. The definition is identical to the **DIHEdral** definition (see also Eq. 4.6), except for wildcards. The wildcard cascading for improper angles is as follows: (a b c d) $\rightarrow$ (a X X d) $\rightarrow$ (X b c d) $\rightarrow$ (X b c X) $\rightarrow$ (X X c d). The program automatically performs the interchange (a b c d) $\rightarrow$ (d c b a) where this is required.

IMPRoper <**selection**> <**selection**> <**selection**> <**selection**> [**MULT** <**integer**>] { <**real**> <**integer**> <**real**> } is an atom-based version of the IMPRoper statement. The definition of the real and integer numbers is identical to the DIHEdral definition (see also Eq. 4.6). The statement can apply to more than one improper angle, depending on the number of bonds that match the quadruple atom selection. The energy constants, periodicities, and offsets can be set to the string TOKEn, which implies that the corresponding constant will be untouched by this statement.

LEARn <**learn-statement**> **END** learns atom-based parameters from one or more sets of Cartesian coordinates (see Section 3.4).

NBFIx <**type**> <**type**> <**real**> <**real**> <**real**> <**real**> adds a Lennard-Jones parameter set for the specified pair of atom types to the parameter database. The first two real numbers are the A, B coefficients (Eqs. 4.12 and 4.13) for all nonbonded interactions except the special 1–4

interactions; the second pair of reals is for the 1–4 (NBXMod=±5) nonbonded interactions. Appropriate **NONB** statements have to be specified for both atom types before invoking this statement. The NBFIx statement allows one to deviate from the standard combination rule for the Lennard-Jones potential (Eq. 4.14). Once an NBFIx statement has been issued for a single atom type, say "NBFIx HT HT", the combination rules for subsequent NBONded statements will be affected, i.e., the "NBFIx HT HT" statement is equivalent to a "NONBonded HT" statement.

NBFIx <**selection**> <**selection**> <**real**> <**real**> <**real**> <**real**> is an atom-based version of the NBFIx statement that adds a Lennard-Jones parameter set for the selected pairs of atoms to the atom-based parameters. The definition of the reals is identical to the type-based NBFIx statement. Appropriate atom-based **NONB** statements have to be specified for both atom selections before invoking this statement, e.g.,

```
NONB ( chemical C* )  0.12  3.74     0.12  3.74
NONB ( chemical N* )  0.24  2.85     0.24  2.85
NBFIx ( chemical C* ) ( chemical N* )  10. 1000. 10. 1000.
```

Note that the TOKEn keyword is not allowed for the reals. Once an NBFIx statement has been issued for a single atom type, say "NBFIx (type HT) (type HT)", the combination rules for subsequent NBONded statements will be affected, i.e., the "NBFIx (type HT) (type HT)" statement is equivalent to a "NONBonded (type HT)" statement.

NBONds { <**nbonds-statement**> } **END** applies to both electrostatic and van der Waals energy calculations. It sets up global parameters for the nonbonded interaction list generation and determines the form of subsequent nonbonded energy calculations (see Eq. 4.8).

NONB <**type**> <**real**> <**real**> <**real**> <**real**> adds a Lennard-Jones parameter set for pairs of atoms of the same specified type to the parameter database. The first pair of reals is ϵ,σ (Eq. 4.8) for all nonbonded interactions except the special 1–4 interactions; the second pair is ϵ,σ for the 1–4 nonbonded interactions (NBXMod=±5).

NONB <**selection**> <**real**> <**real**> <**real**> <**real**> is an atom-based version of the NONB statement that adds a Lennard-Jones parameter set for the selected atoms to the atom-based parameters. The definition of the reals is identical to the type-based NONB statement. Note that the TOKEn keyword is not allowed for the reals.

REDUce { <**reduce-statement**> } **END** derives type-based parameters from existing atom-based parameters (see Section 3.5).

RESEt [**ALL** | **TYPE** | **ATOM**] erases all information about type- and atom-based parameters. The optional specification of ALL erases

both databases, TYPE erases just the type base, and ATOM erases just the atom base (default: ALL).

VERBose produces a verbose listing of all atom-based parameters.

<nbonds-statement>:==

BSHForce specifies beta for electrostatic force shifting (Eq. 4.16). Should be used in combination with the SHFOrce statement (default: ONE).

CDIE|RDIE specifies exclusive flags: constant dielectric (Coulomb's law) or $1/r$-dependent dielectric (Eq. 4.16). RDIE may only be used in combination with VSWItch, SWITch, and REPEl=0. CDIE may be used in combination with VSWItch, SHIFt or SHFOrce, and REPEl=0 or in combination with TRUNcation and REPEl=0 (default: CDIE).

CTOFNB=<real> specifies the distance r_{off} at which the switching function or shifting function forces the nonbonded energy to zero (Eqs. 4.8, 4.16) (default: 7.5 Å).

CTONNB=<real> specifies the distance r_{on} at which the switching function becomes effective (Eq. 4.8) (default: 6.5 Å).

CUTNb=<real> specifies the nonbonded interaction cutoff r_{cut} for the nonbonded list generation (default: 8.5 Å).

E14Fac=<real> specifies the factor e_{14} for the special 1–4 electrostatic interactions (Eq. 4.17) (default: 1.0).

EPS=<real> specifies the dielectric constant ϵ (Eq. 4.16) (default: 1.0).

GROUp | ATOM specifies exclusive flags: group by group or atom by atom cutoff for nonbonded list generation (default: ATOM).

INHIbit=<real> specifies the distance $d_{inhibit}$ (Eq. 4.7) between two atoms below which the van der Waals potential (Eq. 4.8) is truncated (default: 0.25 Å).

IREXponent=<integer> specifies the exponent *irexp* for the repel function (Eq. 4.8) (default: 2).

NBXMod=$+1|-1|+2|-2|+3|-3|+4|-4|+5|-5$ Exclusion list options:

+−1 no nonbonded exclusions, that is, all nonbonded interactions are computed regardless of covalent bonds.

+−2 excludes nonbonded interactions between bonded atoms.

+−3 excludes nonbonded interactions between bonded atoms and atoms that are bonded to a common third atom. This definition is purely based on the list of covalent bonds and is not affected by the list of bond angles.

+−4 excludes nonbonded interactions between bonded atoms, atoms that are bonded to a common third atom, and or atoms that are connected to each other through three bonds. This definition is purely based on the list of covalent bonds and is not affected by the list of bond angles, dihedral or improper angles.

+−5 same as (+-3), but the 1–4 nonbonded interactions are computed using the 1–4 Lennard-Jones parameters and the electrostatic scale factor e_{14} (Eqs. 4.17 and 4.18). This definition is purely based on the list of covalent bonds and is not affected by the list of bond angles, dihedral or improper angles.

A positive mode value causes explicit nonbonded exclusions (see exclusion statement, Section 3.1.1) to be taken into account; a negative value causes them to be discarded (default: 5).

RCONst=<real> specifies the energy constant C_{rep} for the repel function (Eq. 4.8) (default: 100.0).

REPEl=<real> specifies k_{rep}: if > 0, this option turns on the repel function (Eq. 4.8) and turns off the electrostatic energy. k_{rep} specifies the factor by which to multiply the van der Waals radius R_{min} (default: 0).

REXPonent=<integer> specifies the exponent $rexp$ for the repel function (Eq. 4.8) (default: 2).

SWItch|SHIFt|SHFOrce specifies exclusive flags: electrostatic switching or potential (SHIF) or force (SHFO) shifting. SWITch may only be used in combination with RDIE, VSWItch, and REPEl=0. SHIFt or SHFOrce may only be used in combination with CDIE, VSWItch, and REPEl=0 (default: SHIFt).

TOLErance=<real> specifies the distance that any atom is allowed to move before the nonbonded list gets updated. Note: if switching or shifting functions are used, the program expects CUTNB $\geq$ CTOFNB + 2TOLErance. In this way the nonbonded energy is independent of the update frequency. For the REPEl option, CUTNB and TOLErance should be chosen such that CUTNB $\geq r_{\max} + 2$TOLErance, where $r_{\max}$ is the maximum van der Waals radius. TOLErance has no influence on the TRUNcation option. (default: 0.5 Å).

TRUNcation turns off switching or shifting; i.e., the nonbonded energy functions are "truncated" at CUTNb regardless of the values of CTONNB and CTOFNB. All nonbonded energy terms that are included in the current nonbonded list are computed. May only be used in combination with CDIE. Note: in general, the nonbonded energy will not be conserved before and after nonbonded list updates when using TRUNcation. (default: inactive).

VSWItch turns on van der Waals switching. May only be used in

combination with RDIE, SWITch, and REPEl=0 or in combination with CDIE, SHIFt or SHFOrce, and REPEl=0 (default: active).

WMIN=<real> specifies the threshold distance for close contact warnings, i.e., a warning is issued when a pair of atoms gets closer than this distance unless the nonbonded interaction is excluded by the NBXMod option (default: 1.5 Å).

<hbonds-statement>:==

AAEX=<real> specifies the exponential for the angle term between the hydrogen, acceptor and acceptor antecedent atoms n (Eq. 4.25) (default: 2).

ACCEptor=<logical> is a flag indicating whether to include the acceptor antecedent angular term in Eq. 4.25 (default: TRUE).

ACUT=<real> specifies the angular cutoff for the angle between acceptor, hydrogen and donor atoms θ_{A-H-D} (Eq. 4.25). 0° corresponds to a linear hydrogen bond (default: 100°).

AEXP=<real> specifies the attractive exponential j (Eq. 4.25) (default: 4).

AON=<real> specifies $\theta_{h_{on}}$, a switching function parameter for the hydrogen bond angle (Eq. 4.25) (default: 60.0°).

AOFF=<real> specifies $\theta_{h_{off}}$, a switching function parameter for the hydrogen bond angle (Eq. 4.25) (default: 80.0°).

DCUT=<real> specifies r_{AD}, the heavy atom donor to heavy atom acceptor cutoff (Eq. 4.25) (default: 7.5).

DOFF=<real> specifies $r_{h_{off}}$, a switching function parameter for the hydrogen bond distance (Eq. 4.25) (default: 6.5).

DON=<real> specifies $r_{h_{on}}$, a switching function parameter for the hydrogen bond distance (Eq. 4.25) (default: 5.5).

HAEX=<real> specifies the exponential for the angle term between the donor, hydrogen, and acceptor atoms, m (Eq. 4.25) (default: 4).

REXP=<real> specifies the repulsive exponential i (Eq. 4.25) (default: 6).

TOLErance=<real> specifies how far atoms are allowed to move before the hydrogen-bond list gets updated. Note: the program expects DCUT $\geq$ DOFF + 2TOLErance. In this way the hydrogen-bonded energy is independent of the update frequency (default: 0.5).

3.2.2 Requirements

The atom selections of atom-based parameter statements are partially fragile (Section 2.15); i.e, they are mapped when the molecular structure is modified, but newly created atoms do not have atom-based parameters associated with them.

3.3 Writing a Parameter File

The write parameter statement writes the current parameter database to the specified output file. Normally, only the type-based parameters are written.

3.3.1 Syntax

WRITe PARAmeter { <**write-parameter-statement**> } **END** is to be invoked from the main level of X-PLOR. The END statement activates the write process.

<**write-parameter-statement**>:==

OUTPut=<**filename**> specifies the filename (default: OUTPut).

3.3.2 Example of Type-based Parameters

The following example is a partial list of the parameter file "param19.pro" for proteins:

```
bond C    C      450.0  1.38
bond C    CH1E   405.0  1.52
bond C    CH2E   405.0  1.52

angle C    C    C       70.0 106.5
angle C    C    CH2E    65.0 126.5

dihedral CR1E C    C    C    100.0  2 180.0
dihedral CR1E C    C    NH1  100.0  2 180.0
dihedral X    C    CH1E X    0.0    3   0.0
dihedral X    C    CH2E X    0.0    3   0.0

improper C    OC   OC   CH2E 100.0  0   0.0
improper C    X    X    C     25.0  0   0.0

nbonds
  NBXMOD=5  ATOM CDIEL SHIFT vswitch
  CUTNB=8.0  CTOFNB=7.5  CTONNB=6.5  EPS=1.0  E14FAC=0.4  WMIN=1.5
end

NONBonded  H   0.0498 1.4254   0.0498 1.4254
NONBonded  C   0.1200 3.7418   0.1000 3.3854
```

3.3.3 How the Type-based Parameter Retrieval Works

This example shows how X-PLOR finds parameters for the covalent bonds, bond angles, improper angles, and nonbonded energy terms. It demonstrates that the introduction of atom types greatly reduces the number of parameters that have to be specified. The parameter retrieval is carried out when the energy statement is issued.

```
topology
   residue test
   atom a1  type=b  end
   atom a2  type=b  end
   atom a3  type=b  end
```

```
   atom a4  type=a  end
   atom a5  type=a  end
   atom a6  type=a  end

   bond a6    a5
   bond a5    a4
   bond a4    a2
   bond a4    a3
   bond a4    a1

   angle a6 a5 a4                    {* One could have saved work here by *}
   angle a5 a4 a3                    {* using the autogenerate option.    *}
   angle a5 a4 a2
   angle a5 a4 a1
   angle a1 a4 a3
   angle a1 a4 a2
   angle a2 a4 a3

   dihedral a6 a5 a4 a3
   dihedral a6 a5 a4 a2
   dihedral a6 a5 a4 a1

   improper a1 a2 a3 a4
   end
end

parameter
   bond a  b  100.0       1.0
   bond a  a  100.0       1.0
   angle a  a  a   50.0  100.0
   angle b  a  b   50.0  100.0
   angle a  a  b   50.0  100.0
   dihedral a  a  a  b   80.0  3 0.0
   improper b  b  b  a   30.0  0 180.0
   nonbonded a 0.1 1.0 0.1 1.0
   nonbonded b 0.1 1.0 0.1 1.0
end

segment
  name=test
  molecule number=1 name=test end
end

energy end
```

3.3.4 Examples of Atom-based Modifications of the Parameters

The first example shows how to set the energy constants of all improper angles involving tyrosine residues to 500 kcal mol^{-1} rad^{-2}. The periodicity and the equilibrium constants are untouched by the TOKEn keyword.

```
parameter
   improper ( resname tyr ) ( resname tyr )
            ( resname tyr ) ( resname tyr )  500 TOKEN TOKEN
end
```

In the next example, the energy constants of all bonds are set to 400 kcal $mole^{-2}$ $Å^{-2}$:

```
parameter
   bond ( all ) ( all ) 400. TOKEN
end
```

In the next example, the energy constant for the ω torsion angle for X-Pro peptide bonds is set to 500 kcal mole^{-1} Å^{-2}. This example applies only to parameter files that define the ω torsion angle as a dihedral angle, such as "toph19.pro" or "tophcsdx.pro".

```
parameter
   dihedral ( name CA and resname * ) ( name C and resname * )
            ( name N and resname PRO ) ( name CA and resname PRO )
            500 TOKEN TOKEN
end
```

The next example is similar to the previous one, except that, in addition to increasing the energy constant, the periodicity is set to one and the equilibrium angle is set to 0°; i.e., only *trans* prolines are allowed.

```
parameter
   dihedral ( name CA and resname * ) ( name C and resname * )
            ( name N and resname PRO ) ( name CA and resname PRO )
            500 1 0.
end
```

The last example shows how to allow both *cis* and *trans* conformations for the peptide bond between residues 40 and 41 regardless of amino-acid type:

```
parameter
   dihedral ( name ca and resid 40 ) ( name c and resid 40 )
            ( name n and resid 41 ) ( name ca and resid 41 )
   1250. 2.  180.
end
```

3.4 Learning Atom-based Parameters

This facility can be used to obtain bond, bond angle, dihedral angle, or improper angle equilibrium parameters and energy constants from selected atoms of Cartesian coordinate sets. The coordinates are specified in the main coordinate set. The statement learns parameters only for those interaction terms that are turned on by the flags statement (Section 4.5). The learned parameters will take precedence over the type-based parameters.

The equilibrium geometry parameters can be directly obtained from a single coordinate set or averaged over successive coordinate sets. If just a single coordinate set is available, one can learn only the equilibrium geometry, not the energy constants. If an ensemble of coordinates is available, energy constants can be derived assuming equipartition of energy among the different internal coordinates. This is only approximately verified in a real system, since there is actually coupling among the internal coordinates.

$$k_b = \frac{kT}{2 < (r - < r >)^2 >} \tag{3.1}$$

$$k_\theta = \frac{kT}{2 < (\theta - < \theta >)^2 >}$$
$$k_\phi = \frac{kT}{2 < (\phi - < \phi >)^2 >} \quad (3.2)$$

The brackets represent an average over the ensemble of coordinate sets. $kT/2$ is the mean thermal energy per harmonic degree of freedom at $T = 298K$; Eqs. 4.4, 4.5, and 4.6 define the other constants. The last expression assumes that all dihedral angles and torsion angles are represented by a harmonic functional form with periodicity n set to zero (Eq. 4.6). In fact, the learn facility will set the periodicity of all "learned" dihedral and improper angles to zero. In the case that one of the variances in the denominators of Eq. 3.1 becomes zero, the corresponding energy constant is set to 999999.

Parameter learning is not possible for nonbonded and hydrogen-bonded parameters.

<**learn-statement**>:==

INITiate { <**learn-options**> } initializes selected parameters for the learning process.

ACCUmulate includes data from the current main coordinate set in the running averages.

TERMinate calculates final averages.

<**learn-options**>:==

SELEction=<**selection**> learns atom-based parameters for specified atoms. All atoms of a particular interaction term (bond, bond angle, dihedral angle, improper angle) have to be selected in order for the parameters for that term to be learned (default: (ALL)).

MODE=STATistics | **NOSTatistics** specifies whether energy constants are to be learned using Eq. 3.1. The energy constants will be learned if STATistics is specified; otherwise the energy constants will be untouched (default: STATistics).

Note: It is important that the learn options be specified in the initialization stage, e.g.,

```
1 learn initiate selection=( name c* ) MODE=STATistics end
2 learn accumulate end
3 learn terminate  end
```

3.4.1 Requirements

The atom selection is fragile (Section 2.15).

3.4.2 Example: Learning Unknown Equilibrium Parameters from Coordinates

In the following example, a protein and ligand are considered. The molecular structure of the the protein and the ligand have to be generated as outlined in Sections 3.7 and 3.13. The ligand requires the definition of the topology, e.g.,

```
topology
   autogenerate angles=true dihedrals=false end
   residue LIGA
      atom A type=C end
      atom B type=C end
      atom C type=C end
      atom D type=O end
      bond A B
      bond B C
      bond C D
   improper A B C D
   end
end

segment
   name=LIGA
   molecule number=1 name=LIGA end
end
```

Note that mass statements may be required if the atom types of the ligand are non-standard.

The protein parameters can be obtained from one of X-PLOR's protein parameter files. In general, the ligand parameters will be unknown. Suppose that the ligand coordinates are known from an appropriate crystal structure. We can learn the unknown ligand parameters from the known Cartesian coordinates. For purposes of structure determination, it is usually sufficient to set the energy constants to a uniform value.

The following statements define the unknown ligand parameters. They should be inserted in all X-PLOR protocols at any place after the molecular structure and coordinate files have been read and before the first energy evaluation is performed. These statements must also be inserted just before the hbuild statement in the molecular structure generation protocol (Section 3.13).

```

                 {*Learn bond, bond angle, and improper angle parameters.*}
flags exclude * include bonds angles impropers end
parameters
    learn initiate sele=(segid LIGA) mode=nostatistics end
    learn accumulate end
    learn terminate end
end

                                                {*Set the energy constants.*}
parameters
   BOND   (segid LIGA) (segid LIGA) 500. TOKEN
   ANGLE  (segid LIGA) (segid LIGA) (segid LIGA) 500. TOKEN
   IMPR  (segid LIGA) (segid LIGA) (segid LIGA) (segid LIGA) 500. TOKEN TOKEN

                         {*Set the nonbonded parameters (only if required).*}
```

```
17    NBON ( (name A or name B or name C) and segid "LIGA" )  0.1 3.5 0.1 3.5
18    NBON ( name D and segid "LIGA" )  0.1 3.4 0.1 3.4
19 end
20
21                                                {* activate other energy terms *}
22 flags include vdw elec pvdw pele end.
```

Note that the learn statement automatically sets the periodicity of all learned dihedral and improper angles to zero. Also note that the user has to specify improper and dihedral angles in the topology definition of the ligand (see above) in order to maintain planarity and chirality in certain parts of the ligand. Nonbonded parameters may have to be set by appropriate parameter statements unless they are already defined through type-based parameters. Finally, one has to activate the nonbonded energy terms and any other energy terms that might be needed, using the flags statement.

This procedure works fine unless there are unknown hydrogen positions for the ligand. In this case, the selection in the learn statement must exclude the unknown atoms. Furthermore, the parameters for terms involving the hydrogens must be set explicitly. An example is provided in Section 3.13.5.

3.4.3 Example: Learning Atom-based Parameters from an Ensemble of Structures

The learn statement is used to derive equilibrium geometries and energy constants simultaneously from a thermal ensemble of ten coordinate files:

```
1
2         {*Only the active energy terms are affected by the learn statement.*}
3 flags exclude * include bonds angles dihedrals impropers end
4
5                                               {*Initiate the learning process.*}
6 parameters
7    learn initiate sele=(all) mode=statistics end
8 end
9
10                                    {*Loop through the ensemble of coordinates.*}
11 for $filename in ( "a1.pdb" "a2.pdb" "a3.pdb" "a4.pdb" "a5.pdb"
12                    "a6.pdb" "a7.pdb" "a8.pdb" "a9.pdb" "a10.pdb" )
13     loop main
14
15     coordinates @@$filename
16     parameters
17        learn accumulate end
18     end
19 end loop main
20
21                                       {*Now we terminate the learning process.*}
22 parameters
23    learn terminate end
24 end
25
26            {*One could now compute energies with the learned parameters or *}
27            {*reduce them to type-based parameters and write them to a file.*}
```

Instead of learning from an ensemble of coordinates, one could have also learned the coordinates from a trajectory (Section 11.2).

3.4.4 Example: Learning a New Set of Equilibrium Parameters for DNA

In the following example the learn statement is used to derive equilibrium geometries for oligonucleotides. In this specific example, a B-type DNA structure is used. The energy constants are set to generic values.

```
remarks  file   parameter/learn.inp
remarks  learning equilibrium parameters from an ideal B-DNA structure

{===>}
structure @generateb.psf end       {* Read structure file and coordinates *}
coordinates @generateb.pdb         {* for a B-type oligonucleotide.       *}

{===>}
parameter
    @TOPPAR:param11.dna       {* Read the original set of DNA parameters. *}
end

   parameter
                                        {* Learn equilibrium parameters. *}
      learn init selection=(  resid 1:16 ) mode=nostat end
      learn accu end
      learn term end

                           {* Set all energy constants to generic values. *}
      bonds ( resid 1:16 )
            ( resid 1:16 ) 500. TOKEN

      angles ( resid 1:16 )
             ( resid 1:16 )
             ( resid 1:16 ) 500. TOKEN

      improper ( resid 1:16 )
               ( resid 1:16 )
               ( resid 1:16 )
               ( resid 1:16 )   500. TOKEN TOKEN
      dihedral ( resid 1:16 )
               ( resid 1:16 )
               ( resid 1:16 )
               ( resid 1:16 )   0. TOKEN TOKEN

          {* Reduce the atom-based parameters to the chemical type base. *}
      reduce selection=(  resid 1:16  )
             overwrite=true
             mode=nostat
      end

   end

               {* Write modified parameters to the file "param11bx.dna". *}
   set display=param11bx.dna end
   display set message=off echo=off end
   write parameters output=param11bx.dna  end
   open param11bx.dna access=append end
   display set message=on echo=on end

stop

```

3.5 Reducing to Type-based Parameters

The reduce statement allows one to reduce the current parameters (atom-based as well as type-based) to type-based parameters using the CHEMical atom properties (Section 4.5). Each type-based equilibrium constant is obtained as an average of all instances of that type of interaction (bond, bond angle, dihedral angle, or improper angle) in the molecular structure. The energy constants can be obtained as a simple average or from the same statistical analysis (Eq. 3.1) as that used for the learn statement. The latter option is useful only when several instances of the same type-based interaction occur in the molecular structure. If there is only one instance or the variance is zero, the "reduced" energy constant is set to 999999. The periodicities (n) of dihedral and improper angles have to be consistent for all instances of atom-based parameters; otherwise a warning message will occur. The "reduced" periodicities are taken directly from the atom-based parameters. The reduce statement affects parameters only for those interaction terms that are turned on by the flags statement (Section 4.5).

3.5.1 Syntax

<reduce-statement>:==

SELEction=<selection> reduces parameters for specified atoms. All atoms of a particular interaction term (bond, bond angle, etc.) have to be selected in order for the parameters for that term to be learned (default: (ALL)).

OVERwrite=TRUE | FALSE will overwrite the existing REDUce statement type-based parameters if TRUE. If FALSE, the REDUce statement will add only new type-based parameters; it will not overwrite existing ones (default: FALSE).

MODE=AVERage | STATistical obtains energy constants from a simple average (AVERage) or from the statistical analysis described in Eq. 3.1 (STATistical). The latter option is useful only when several instances of the same type-based interaction occur in the molecular structure. For single instances or zero variance, the energy constant will be set to 999999 (default: STATistics).

3.5.2 Example: Reducing Parameters Learned Previously

Suppose one has learned atom-based parameters for a ligand with segment identifier LIGA from Cartesian coordinates. These atom-based parameters are reduced to type-based parameters and then written to a file:

```
1 parameters reduce sele=(segid LIGA ) mode=average end end
2 write parameters output=newparameters.pro end
```

For each bonded pair of chemical types, an equilibrium geometry is obtained by averaging the atom-based parameters of all the bonds in the molecular structure that match this chemical type pair. For example, in a protein, this could imply averaging over all $C-C$ bonds in aromatic rings, or all $C-O$ carbonyl bonds. The same averaging is done for bond angles, dihedrals and impropers, equilibrium geometries, and energy constants. Dihedral and improper periodicities will correspond to their atom-based values. The new parameter file "newparameters.pro" can be read by X-PLOR in subsequent runs.

3.6 Topology and Parameter Files

This section describes the most important parameter and topology files that are distributed with X-PLOR. A pair of parameter and topology files represents a force field or empirical energy function. Force fields are supplied for proteins, nucleic acids, carbohydrates, and lipids, as well as a number of other biological molecules. All parameter files represent polar hydrogens (i.e., hydrogens that can form hydrogen bonds) explicitly, whereas aliphatic hydrogens are either explicitly included or implicitly modeled by appropriate modification of the heavy-atom van der Waals parameters. A number of specialized force fields are available for X-ray crystallographic or NMR-spectroscopic structure determination. All parameter and topology files can be found in the "toppar" directory.

3.6.1 CHARMM "top_all22*" and "par_all22*" All Hydrogen Force Field

This new CHARMM force field (MacKerell et al., in preparation) contains parameters for proteins, nucleic acids, and lipids. All hydrogens are explicitly included.

3.6.2 CHARMM "toph19.pro" and "param19.pro" Files for Proteins (Explicit Polar Hydrogens)

This force field, as described by Brooks et al. (1983), treats hydrogen implicitly by modification of the parameter of the hydrogen antecedent, except for polar hydrogens such as $N-H$. In the latter case, the hydrogens are treated explicitly in order to obtain a more realistic hydrogen bond potential. This force field does not use the explicit hydrogen bonded potential (Eq. 4.25); i.e., the HBONds energy term will be zero. The hydrogen bond energy is part of the ELEC and VDW energy terms. However, the DONOr and ACCEptor statements (Section 3.1.1) are still present in order to allow the user to analyze hydrogen bonds by using the print statement (Section 5.1.1). The peptide-bond linkages (Section 3.7.3) are defined in "toph19.pep".

3.6.3 CHARMM "toph11.dna" and "param11bx.dna" Files for Nucleic Acids (Explicit Polar Hydrogens)

This force field was derived from the "param11" parameters described by Brooks et al. (1983). Equilibrium values were adjusted to achieve greater self-consistency for B-type DNA structure. The learn feature of the parameter statement was used (Section 3.4.4) to derive the equilibrium parameters. Energy constants are compatible with the "parhcsdx.pro" parameters for proteins. It is often necessary to apply additional sugar pucker, base planarity, and hydrogen bond restraints to maintain the conformation of oligonucleotides (see Section 3.13.7).

3.6.4 Files "topnah1e.dna" and "parnah1e.dna" for Nucleic Acids (Explicit All Hydrogens)

This force field, by Nilsson and Karplus (1986), treats all hydrogens explicitly. It is often necessary to apply sugar pucker, base planarity, and hydrogen bond restraints to maintain the conformation of oligonucleotides (see Section 3.13.7).

3.6.5 AMBER/OPLS "tophopls.pro", "parhopls.pro" Files (Explicit Polar Hydrogens)

This force field is described in Jorgensen and Tirado-Rives (1988) and Weiner et al. (1984). The peptide-bond linkages (Section 3.7.3) are defined in "toph19.pep". The TIP3p solvent model is included in the "tophopls.pro", "parhopls.pro" files.

3.6.6 Files "toph19.sol" and "param19.sol" for Water (TIP3p Model)

This force field describes the TIP3p water model (Jorgensen et al. 1983). The parameter files are intended to be used in combination with the "toph19.pro"force field for proteins. Combination with other force fields for proteins and nucleic acids are possible except in the case of the AMBER/OPLS force field which already contains water parameters. The "param19.sol" file also contains special adjustments for solvent-solute interactions.

3.6.7 Files "toph3.cho" and "param3.cho" for Carbohydrates

This force field is intended for crystallographic structure determination. It may be used in conjunction with the "parhcsdx.pro" and "tophcsdx.pro" files for proteins. The parameters originated from Ha et al. (1988) and were modified as described by Weis et al. (1990).

3.6.8 Files "toph19.chromo" and "param19.chromo" for Chromophores

This force field was developed by Treutlein et al. (1992) for molecular dynamics simulations of the bacterial reaction center.

3.6.9 Files "parhcsdx.pro" and "tophcsdx.pro" for Crystallographic Refinement (Polar Hydrogens)

This force field was developed by Engh and Huber (1991). It is a replacement for the "toph19x.pro" and "param19x.pro" files that were used in previous versions of X-PLOR. Inconsistencies for a number of residues have been removed, which results in improved geometry and R values. The energy constants were derived from observed uncertainties in geometric parameters obtained from the Cambridge Crystallographic Database for small model compounds. This file also contains patches to change the protonation degree of histidine residues (see also Section 3.8.4). The peptide-bond linkages (Section 3.7.3) are defined in "toph19.pep".

It was necessary to modify a number of improper and dihedral parameters compared to the "toph19.pro" and "param19.pro" files. The twofold dihedral angle term for peptide bonds was replaced by a onefold term with a large energy constant. When using the original twofold term, transitions of peptide bonds from *trans* to *cis* conformations occurred at high simulated annealing temperatures. In the case of X-Pro peptide bonds, the twofold term was retained, but with an increased energy constant. To facilitate transitions between *trans* and *cis* prolines, the user should lower this energy constant with the atom-based parameter statement:

```
parameter
   dihedral ( name CA and resname * ) ( name C and resname * )
            ( name N and resname PRO ) ( name CA and resname PRO )
            5. 2 180.
end
```

The energy constants of the improper torsion angle that specify the chirality of C^{α} carbon atoms were also increased. The energy constants of the improper torsion angles that specify the planarity of aromatic rings were increased. In addition, the 1–4 and 1–5 nonbonded interactions in aromatic rings were included in the calculation. Without the increased energy constants and the nonbonded interactions, aromatic rings collapsed in a few cases where the initial coordinates contained bad contacts or large strain.

For crystallographic refinement, it is reasonable to reduce or delete the charges on charged residues, such as Lys, Arg, or Glu. To modify the charges, one can use the vector statement (Section 2.16); e.g.,

```
vector do ( charge=0.0 ) ( resname LYS and
    ( name ce or name nz or name hz* ) )       {* Turn off charges for LYS.*}
vector do ( charge=0.0 ) ( resname GLU and
    ( name cg or name cd or name oe* ) )       {* Turn off charges for GLU.*}
```

```
vector do ( charge=0.0 ) ( resname ASP and
    ( name cb or name cg or name od* ) )       {* Turn off charges for ASP.*}
vector do ( charge=0.0 ) ( resname ARG and
    ( name cd or name *E or name cz or name NH* or name HH* )
```

3.6.10 Files "parallhdg.pro" and "topallhdg.pro" for NMR Structure Determination of Proteins (All Hydrogens)

The weighting among bonds, bond angles, planarity, and chirality reflects the special requirements when using interproton distance restraints. Special improper terms have been added to maintain the *trans*- or *cis*-peptide bonds, chirality of all tetrahedral centers, and planarity of peptide bonds and aromatic rings. The peptide bond conformation is described by an improper angle that constrains the conformation to *trans*. The patch CISP allows the user to switch a *trans*- to a *cis*-peptide bond. The patch LTOD allows the user to switch from L to D amino acids. The peptide-bond linkages (Section 3.7.3) are defined in "toph19.pep".

Van der Waals radii

The σ values (cf. Eqs. 4.8 and 4.10) in the parameter file "parallhdg.pro" are different from those used in the other all-hydrogen force fields. The changes were made to make the van der Waals radii more consistent with those used by other NMR refinement programs. The σ values have been set such that the van der Waals radii will be exactly like those used by the DISGEO program (Havel and Wüthrich 1984) when REPEl (Section 3.2.1) is set to 0.8. In addition, van der Waals radii of potential hydrogen bond donor–acceptor pairs have been reduced by appropriate NBFIx statements. The actual van der Waals radii used for different values of REPEl are given in the table below (in Å).

type	repel=1.0	repel=0.9	repel=0.8	repel=0.75
H	1.250	1.125	1.0	0.938
C	1.875	1.688	1.5	1.406
N	1.688	1.519	1.350	1.266
O	1.558	1.402	1.246	1.168
S	2.102	1.892	1.682	1.577

It should be noted that the van der Waals radii in "parallhdg.pro" are not appropriate with the standard Lennard-Jones potential (Eq. 4.8).

Convention for Hydrogen Names

The hydrogen names follow the scheme that is adopted by the CHARMM all-hydrogen force fields (Brooks et al. 1983). If a different scheme is required,

the user can either rename the hydrogens in the topology file or change the molecular structure after it has been created. An example of how to change the hydrogen naming to the convention used by the DIANA program is supplied in the file "xplortodiana.inp" in the "toppar" directory.

3.6.11 Files "parallhdg.dna" and "topallhdg.dna" for NMR Structure Determination of Nucleic Acids (All Hydrogens)

The weighting among bonds, bond angles, planarity, and chirality reflects the special requirements when using interproton distance restraints. It is often necessary to apply sugar pucker, base planarity, and hydrogen bond restraints to maintain the conformation of oligonucleotides (see Section 3.13.7).

3.7 Generating the Molecular Structure

The segment statement generates the molecular structure by interpreting the coordinate file to obtain the residue sequence or by explicitly specifying the residue sequence. The residues have to be defined in the topology statement (Section 3.1.1). A segment is defined as a protein with one continuous polypeptide chain, a continuous polypeptide chain itself, a substrate, a cofactor, a ligand, a set of water molecules, or a single nucleic acid strand. For a combination of proteins, water, substrates, ligands, and/or nucleic acids, the user should first split the original coordinate file into several files containing the segments, generate each segment separately, and, if necessary, apply appropriate patches to establish covalent links between segments or within segments, such as disulfide bridges.

3.7.1 Syntax

<**residue-number**> specifies the number of a residue, a four-character string that can include nonnumerical characters.

<**segment-name**> specifies the name of a segment, a four-character string that can include nonnumerical characters.

SEGMent { <**segment-statement**> } **END** is invoked from the main level of X-PLOR.

<**segment-statement**>:==

CHAIn { [<**chain-statement**>] } **END** generates a sequence of residues. The END statement activates the generation.

MOLEcule NAME=<**residue-name**> **NUMBer=**<**integer**> **END** generates individual molecules such as in liquids (default of NUMBer is 1). The residue numbers are assigned sequentially, starting with 1.

The residue name has to be defined in the topology statement (see Section 3.1.1). The END statement activates the generation of the molecules.

NAME=<segment-name> specifies the segment name of all atoms to be generated using one of the following statements (default is " ").

<chain-statement>:==

COORdinates { <pdb-record> } END takes a sequence from a coordinate file. It interprets the atom records of a PDB coordinate file in terms of the sequence of the molecule. The residue numbers are taken directly from the PDB coordinate file; they do not have to be sequential. Note that this statement does not actually read the x,y,z coordinates. One should use the coordinate statement (Section 6.1) to read the coordinates. The segment name has to match characters 73 through 76 in the PDB coordinate file. (For the definition of <pdb-record>, see Section 6.1.)

FIRSt <residue-name>
TAIL=<patch-character>=<*residue-name*> END adds a special patch for the first residue in a sequence to the chain database. In the case of multiple statements, equality has priority before wildcard matching; i.e., the more special assignments come first. The chain database is reset each time the chain statement is invoked.

LAST <residue-name>
HEAD=<patch-character>=<*residue-name*> END adds a special patch for the last residue in a sequence to the chain database. In the case of multiple statements, equality has priority before wildcard matching; i.e., the more special assignments come first. The chain database is reset each time the chain statement is invoked.

LINK <residue-name>
HEAD=<patch-character>=<*residue-name*>
TAIL=<patch-character>=<*residue-name*> END adds a special linkage patch to the chain database. The statement will automatically connect residue i to residue $i+1$; e.g., it creates a peptide linkage. Wildcards are allowed for residue name. In the case of multiple statements, equality has priority before wildcard matching; i.e., the more special assignments come first. The chain database is reset each time the chain statement is invoked.

SEQUence { <residue-name> } END takes the sequence as specified between the SEQUence and the END word. The residue numbers are assigned sequentially, starting with 1.

3.7.2 Requirements

The molecular topology has to be defined (Section 3.1.1). Execution of the segment statement destroys fragile atom selections (Section 2.15).

3.7.3 Example: A Polypeptide Chain

The following example file shows how to set up a polypeptide segment (Tyr, Ala, Glu, Lys, Ile, Ala), assuming that the topology and parameters have been previously read. For completeness, the definitions of the patch residues PEPT, PROP, NTER, and CTER have been included. Normally, these patch residues are already defined in the topology files.

```
topology
   PRESidue PEPT
     ADD BOND -C +N
     ADD ANGLE -CA -C +N
     ADD ANGLE -O  -C +N
     ADD ANGLE -C  +N +CA
     ADD ANGLE -C  +N +H
     ADD DIHEdral  -C +N +CA +C
     ADD DIHEdral  -N -CA -C +N
     ADD DIHEdral  -CA -C +N +CA
     ADD IMPRoper  -C -CA +N -O  {planar -C}
     ADD IMPRoper  +N -C +CA +H  {planar +N}
   END

   PRESidue PEPP
     ADD BOND -C +N
     ADD ANGLE -CA -C +N
     ADD ANGLE -O  -C +N
     ADD ANGLE -C  +N +CA
     ADD ANGLE -C  +N +CD
     ADD DIHEdral  -C +N +CA +C
     ADD DIHEdral  -N -CA -C +N
     ADD DIHEdral  -CA -C +N +CA
     ADD IMPRoper  -C -CA +N -O  {planar -C}
     ADD IMPRoper  +N +CA +CD -C  {planar +N}
  END

  PRESidue NTER
    GROUp
    ADD    ATOM +HT1  TYPE=HC   CHARge=0.35  END
    ADD    ATOM +HT2  TYPE=HC   CHARge=0.35  END
    MODIfy ATOM +N    TYPE=NH3  CHARge=-0.30 END
    ADD    ATOM +HT3  TYPE=HC   CHARge=0.35  END
    DELETE ATOM +H                           END
    MODIfy ATOM +CA             CHARge=0.25  END
    ADD BOND +HT1 +N
    ADD BOND +HT2 +N
    ADD BOND +HT3 +N
    ADD ANGLe +HT1  +N    +HT2
    ADD ANGLe +HT2  +N    +HT3
    ADD ANGLe +HT2  +N    +CA
    ADD ANGLe +HT1  +N    +HT3
    ADD ANGLe +HT1  +N    +CA
    ADD ANGLe +HT3  +N    +CA
    ADD DIHEdral +HT2  +N    +CA   +C
    ADD DIHEdral +HT1  +N    +CA   +C
    ADD DIHEdral +HT3  +N    +CA   +C
    ADD DONOr +HT1  +N
    ADD DONOr +HT2  +N
    ADD DONOr +HT3  +N
  END

  PRESidue PROP
    GROUp
    ADD    ATOM +HT1  TYPE=HC   CHARge= 0.35   END
    ADD    ATOM +HT2  TYPE=HC   CHARge= 0.35   END
    MODIfy ATOM +N    TYPE=NH3  CHARge=-0.20   END
    MODIfy ATOM +CD             CHARge= 0.25   END
    MODIfy ATOM +CA             CHARge= 0.25   END
```

```
60     ADD BOND +HT1  +N
61     ADD BOND +HT2  +N
62     ADD ANGLe +HT1  +N    +HT2
63     ADD ANGLe +HT2  +N    +CA
64     ADD ANGLe +HT1  +N    +CD
65     ADD ANGLe +HT1  +N    +CA
66     ADD ANGLe +CD   +N    +HT2
67     ADD DIHEdral +HT2  +N    +CA   +C
68     ADD DIHEdral +HT1  +N    +CA   +C
69     ADD DONOr +HT1  +N
70     ADD DONOr +HT2  +N
71   END
72
73   PRESidue CTER
74     GROUp
75     MODIfy ATOM -C             CHARge= 0.14  END
76     ADD    ATOM -OT1  TYPE=OC  CHARge=-0.57  END
77     ADD    ATOM -OT2  TYPE=OC  CHARge=-0.57  END
78     DELETE ATOM -O                           END
79     ADD BOND -C    -OT1
80     ADD BOND -C    -OT2
81     ADD ANGLe -CA   -C   -OT1
82     ADD ANGLe -CA   -C   -OT2
83     ADD ANGLe -OT1  -C   -OT2
84     ADD DIHEdral -N    -CA    -C   -OT2
85     ADD IMPRoper -C    -CA    -OT2 -OT1
86     ADD ACCEptor -OT1 -C
87     ADD ACCEptor -OT2 -C
88   END
89 end
90
91 segment
92    name="PROT"
93    chain
94       link pept head - * tail + * end
95       first prop tail + pro end     ! special n-ter for PRO
96       first nter tail + *   end
97       last cter head - * end
98       sequence TYR ALA GLU LYS ILE ALA end
99    end
100  end
```

The information about the patch residues is normally included in the topology file. The information about the peptide linkages is also included in the "toph19.pep" file in the "toppar" directory. Wildcards allow one to use the same patch residues ("PEPT") for all combinations of amino acids. The only exception is Pro, which needs a special N-terminal patch. The residues are numbered consecutively, starting with 1. To use a particular numbering for residues, one should use the COOR option to read the sequence from the coordinate file.

3.7.4 Example: A Polypeptide Chain with an Unknown Section

In crystallography, one frequently encounters the situation where a section is unknown but there is evidence that the polypeptide segment is actually connected. Suppose the chain has the sequence Tyr, Ala, Glu, Lys, Ile, Ala, Ala, Ala, Glu, Asp, Gly, Gly, Gly, and two of the innermost alanines are missing. To generate an appropriate molecular structure, one should follow the example below:

```
segment
   name="PROT"
   chain
      link pept head - * tail + * end
      first prop tail + pro end
      first nter tail + *   end           {* Note that the C-terminus is  *}
                                          {* missing.                     *}
      sequence TYR ALA GLU LYS ILE ALA end
   end
   chain
      link pept head - * tail + * end
      last cter head - * end              {* Note that the N-terminus is  *}
                                          {* missing.                     *}
      sequence GLU ASP GLY GLY GLY end
   end
 end
```

In a real case, one might want to read the sequence from the coordinate file, which would require splitting the original coordinate file into two files containing the two known sections of the chain.

3.7.5 Example: Water Molecules

This example shows how to set up 125 water molecules, assuming that the Tip3p water topology and parameters have been previously read:

```
segment
   name="WAT"
   molecule TIP3 number=125 end
 end
```

A box with edge length 15.55 Å consisting of 125 equilibrated water molecules is supplied in one of the test case input files (file "tipstest1.inp" in the "test" directory).

3.7.6 Example: Solvation of a Solute

The atom selection feature in X-PLOR can be used to solvate a solute, such as protein or nucleic acid. Suppose the user has set up coordinates and molecular structure files for the solute ("solute.pdb" and "solute.psf") and for a box of water ("water.pdb" and "water.psf" with SEGId "WAT") that is large enough to enclose the solute. The following X-PLOR script will create a composite coordinate and molecular structure file ("overlay.pdb" and "overlay.psf") deleting all waters that penetrate the protein:

```
structure @solute.psf end
coordinates @solute.pdb
structure @water.psf end
coordinates @water.pdb

delete
   selection=
        (  byres
         (  segid "WAT"
            and ( ( not segid "WAT" and not ( hydrogen ) ) around 2.6 )
        ))
end

```

```
write structure output=overlay.psf end
write coordinates output=overlay.pdb end
```

If the box of water molecules is not large enough, it can be made larger by duplicating and translating the original box (cf. Section 3.10).

3.8 Patching the Molecular Structure

The patch statement refers to a patch residue (PRESidue), which consists of additions of atoms by the add statement, modifications by the modify statement, or deletions by the delete statement. A patch can establish chain linkages, such as peptide bonds, chain termini, disulfide bridges, and covalent links to ligands.

3.8.1 Syntax

PATCh <patch-statement> END is invoked from the main level of X-PLOR. The END statement activates the patching.

<patch-statement>:== <residue-name>
{ **REFErence=NIL** | **<patch-character>** =**<selection>** } patches the specified selections of atoms using the patch residue with name residue name. The patch character corresponds to the first character in the PRES specification. The specification of NIL implies that in the PRESidue the patch characters are omitted. Multiple references are used to establish links to more than one residue.

3.8.2 Requirements

The molecular structure has to be well defined. Execution of the patch statement destroys fragile atom selections (Section 2.15); e.g., the atom properties (Section 2.16) are conserved during patching, except for the internal stores (STORE1, STORE2, ...). The atom properties of additional atoms are set to default values. Other information, such as information for the xrefin or NOE statements, is lost during patching.

3.8.3 Example: Incorporation of Disulfide Bridges

All protein force fields have the DISU patch residue defined. To make a disulfide bridge, one should use the patch statement; e.g.,

```
topology
   presidue DISU
      group
      modify atom 1CB           charge= 0.19  END
      modify atom 1SG  type=S   charge=-0.19  END
      group
      modify atom 2CB           charge= 0.19  END
      modify atom 2SG  type=S   charge=-0.19  END
```

```
      add bond 1SG 2SG
      add angle  1CB 1SG 2SG
      add angle  1SG 2SG 2CB
      add dihedral   1CA 1CB 1SG 2SG
      add dihedral   1CB 1SG 2SG 2CB
      add dihedral   1SG 2SG 2CB 2CA
   end
end

patch
   reference=1=( resid 15 ) reference=2=( resid 25 )
end
```

will make a disulfide bridge between residue number 15 and residue number 25. (For completeness, the definition of the patch residue DISU is listed here as well. Normally, this patch residue is already included in the topology files.)

3.8.4 Example: Modification of the Protonation Degree of Histidines

The protonation degree of HIS can be changed by using the patch residues HISE and HISD that are defined in file "tophcsdx.pro", which is located in the "toppar" directory. The histidine patches can be invoked by using the patch statement; e.g.,

```
patch HISE
   reference=nil=( resid 14 )
end
```

will change the protonation degree of His-14 to that of a singly HE2-protonated histidine.

3.9 Deleting Atoms

The delete statement removes the selected atoms from the current molecular structure. It will also delete any connections such as bonds, bond angles, or dihedral angles involving an atom that is deleted.

3.9.1 Syntax

DELEte { <**delete-statement**> } **END** is invoked from the main level of X-PLOR. The END statement activates the deletion.

<**delete-statement**>:==

SELEction=<**selection**> selects the atoms that are to be deleted (default: none).

3.9.2 Requirements

The molecular structure has to be well defined. Execution of the delete statement destroys fragile atom selections (Section 2.15); e.g., the atom properties (Section 2.16) are conserved during deletions, except for the internal stores (STORE1, STORE2, ...). Other information, such as information for the xrefin or NOE statements, is lost during deletions.

3.9.3 Example: Delete One Atom

To remove the hydrogen of peptide nitrogen of residue 1, one can specify

```
1 delete
2    selection=(resid 1 and name HN)
3 end
```

3.10 Duplicating the Molecular Structure

The duplicate statement allows one to duplicate the molecular structure or selected atoms of it. The statement duplicates all atom properties, coordinates, connectivities, bonds, angles, etc. It has a renaming feature that allows one to specify a new segment or residue name for the duplicated atoms.

3.10.1 Syntax

DUPLicate { <**duplicate-statement**> } **END** is invoked from the main level of X-PLOR. The END statement activates the deletion.

<**duplicate-statement**>:==

RESIdue=<**residue-name**> specifies the residue name of the duplicated atoms (default: same as original atoms).

SEGId=<**segid-name**> specifies the segment name of the duplicated atoms (default: same as original atoms).

SELEction=<**selection**> selects the atoms that are to be duplicated.

3.10.2 Requirements

The molecular structure has to be well defined. Execution of the duplicate statement destroys fragile atom selections (Section 2.15).

3.10.3 Example: Duplication of Side-Chain Atoms

The following example duplicates the atoms of a side chain. The duplicated atoms are identical to the original ones except for the segment name. The example can be used to set up alternate conformations (see Section 12.8).

```
duplicate
   selection=( resid 40 and not
               ( name ca or name n or name c or name o ) )
   segid="ALT"
end
```

3.11 Structure Statement

The structure statement allows one to read a molecular structure file that has been written previously by the write structure statement. (See Section 3.12 for an explanation of what information is read by the structure statement.)

3.11.1 Syntax

STRUcture { <**structure-statement**> } **END** is invoked from the main level of X-PLOR.

<**structure-statement**>:==

<**psf-records**> adds <psf-records> to the molecular structure database. The <psf-records> contain fixed-field records written by the write structure statement. They are not stored in the rotating statement buffer during loop execution. For the contents of psf-records, see the write structure statement.

RESEt eliminates the current molecular structure in X-PLOR.

3.11.2 Requirements

Execution of the duplicate statement destroys fragile atom selections (Section 2.15).

3.11.3 Example: How to Read a Molecular Structure File

The following example shows how to read the molecular structure file "molecule1.psf":

```
structure
   @molecule1.psf
end
```

3.11.4 Example: Append Two Molecular Structure Files

Molecular structure information can be appended; e.g.,

```
structure
   @molecule1.psf
   @molecule2.psf
end
```

but care should be taken to avoid duplicate atom definitions (e.g., by renaming the SEGId of one molecule with the vector statement).

3.12 Writing a Molecular Structure File

The write structure statement writes the current molecular structure to the specified output file. The molecular structure file (which for historical reasons is sometimes called a PSF—protein structure file) contains information that was generated by the segment statement and modified by the patch statement or the delete statement. Specifically, the molecular structure file contains the following information: atom names, types, charges, and masses; residue names and segment names; and a list of bond terms, angle terms, dihedral terms, improper terms, explicit hydrogen-bonding terms, explicit nonbonded exclusions, and nonbonded group partitions. It does not contain atomic coordinates, parameters, constraints, restraints, or any other information that is specific to effective energy terms, such as diffraction data.

3.12.1 Syntax

WRITe STRUcture { <**write-structure-statement**> } **END** is invoked from the main level of X-PLOR. The END statement activates the write process.

<**write-structure-statement**>:==

OUTPut=<**filename**> specifies the file to which the molecular structure information is to be written.

3.13 Examples for Molecular Structure Generation

The general strategy for setting up the molecular structure for a combination of proteins, water, substrates, ligands, and/or nucleic acids is:

1. Split the original coordinate file into several files containing the segments. A segment is defined as a protein with one continuous polypeptide chain, a continuous protein chain, a substrate, a cofactor, a ligand, all water molecules combined, or a single nucleic acid strand.

2. Generate each segment separately.

3. If necessary, apply appropriate patches to establish covalent links between segments or within segments, such as disulfide bridges.

3.13.1 What to Do about Unknown Atoms

X-PLOR meticulously keeps track of unknown atomic positions and normally terminates execution if one wants to compute energies with unknown atoms. How to get around this? There is a simple answer: create appropriate positions with the graphics, or simply guess coordinates and let the minimizer take care of the geometry. If this is not possible, for instance in cases of disordered side-chain atoms, then the following procedures should be applied: in the molecular structure generation file, ignore the warning messages from the hydrogen builder regarding unknown heavy atom positions. In subsequent files, insert the following statement after reading the molecular structure file and the coordinate file:

```
constraint interaction=( known ) ( known ) end
```

Effective energy term databases may also need some modification; e.g., the crystallographic database should be modified to select only known atoms:

```
xrefin
  selection=( not hydrogen and known )
end
```

Note that X-PLOR occasionally produces unknown atom positions when the molecular dynamics or energy minimization routines run into numerical instabilities. Usually, the energies become very large before this happens. It can occur because the temperature of the molecular dynamics calculation is too high, the time step is too large, or the conformation is extremely strained.

3.13.2 A Standard Protein Structure

```
 remarks  file  generate/generate.inp
 remarks  Generate structure file and hydrogens for a protein

 topology  @TOPPAR:tophcsdx.pro  end                  {*Read topology file.*}

 parameter

@TOPPAR:parhcsdx.pro                                  {*Read empirical potential*}
                                                      {*parameter file CHARMM19 *}
                                                      {*with modifications.     *}

   nbonds                                          {*This statement specifies the  *}
      atom cdie shift eps=1.0  e14fac=0.4          {*nonbonded interaction energy  *}
      cutnb=7.5 ctonnb=6.0 ctofnb=6.5              {*options.  Note the reduced    *}
      nbxmod=5 vswitch                            {*nonbonding cutoff to save some*}
   end                                            {*CPU time.   This statement    *}
                                                  {*overwrites the defaults in    *}
```

```
18                                          {*the parameter file.           *}
19  end
20
21  segment                                                 {*Generate protein.*}
22
23    name="    "                            {*This name has to match the   *}
24                                           {*four characters in columns 73*}
25                                           {*through 76 in the coordinate *}
26                                           {*file; in XPLOR this name is  *}
27                                           {*referred to as SEGId.        *}
28    chain
29      @TOPPAR:toph19.pep                    {*Read peptide bond file;      *}
30      coordinates @amy.pdb                  {*interpret coordinate file to*}
31    end                                     {*obtain the sequence.         *}
32  end
33                                                {*Sometimes different atom*}
34  vector do (name="O") ( name OT1 )             {*names are used.          *}
35  vector do (name="OT") ( name OT2 )
36  vector do (name="CD1") ( name CD and resname ile )
37
38  coordinates @amy.pdb                          {*Here we actually read the*}
39                                                {*coordinates.             *}
40
41                                         {*The generation of cofactors,   *}
42                                         {*waters, etc. would follow here.*}
43                                              {*Note that one has to split*}
44                                              {*the coordinate file into  *}
45                                              {*separate files containing *}
46                                              {*the protein, cofactors,   *}
47                                              {*substrate, water, etc.    *}
48
49                                                    {*Create a S-S bridge.*}
50   patch disu
51     reference=1=( resid 11 )
52     reference=2=( resid 27 )
53   end
54   patch disu
55     reference=1=( resid 45 )
56     reference=2=( resid 73 )
57   end
58
59  flags exclude vdw elec end                {*Do QUICK hydrogen building w/o*}
60                                            {*vdw and elec terms.           *}
61
62  hbuild                                      {*This statement builds       *}
63     selection=( hydrogen )                   {*missing hydrogens, which are*}
64     phistep=45                               {*needed for the force field. *}
65  end
66
67  constraints fix=( not hydrogen ) end      {* Minimize hydrogen positions. *}
68  flags include vdw elec end
69  minimize powell
70     nstep=40
71  end
72  constraints fix=( not all ) end
73
74
75  write coordinates output=generate.pdb end         {*Write out coordinates.*}
76
77  write structure output=generate.psf end        {*Write out structure file.*}
78  stop
```

3.13.3 How to Set Up Unusual Geometries

The following example shows how to set up unusual peptide geometries, such as *D*-amino acids, *cis*-peptide bonds, methylated amino acids, and nonstandard amino acids:

```
remarks  file generate/csa_generate.inp
remarks  Generate structure file and hydrogens for Cylcosporin A

topology
   @TOPPAR:topallhdg.pro                    {*Read standard all-h topology*}
                                            {*file.                       *}
    pres m                                  {*Define methylation patch for*}
                                            {*amides.                     *}
      modify ATOM N    TYPE=NH1 CHARge=-0.360  END
      add atom CN  charge=-0.3 type=CT end
      add atom HN1 charge=0.1 type=HA end
      add atom HN2 charge=0.1 type=HA end
      add atom HN3 charge=0.1 type=HA end
      delete atom HN  end

      bond N CN      bond CN HN1        bond CN HN2        bond CN HN3

      angles N CN HN1       angles N CN HN2        angles N CN HN3
      angles HN1 CN HN2     angles HN1 CN HN3      angles HN2 CN HN3
   end

   RESI ABU                                 {*Define special residue ABU.*}
      ATOM N    TYPE=NH1 CHARge=-0.360 END
      ATOM HN   TYPE=H   CHARge=0.260  END
      ATOM CA   TYPE=CT  CHARge=0.000  END
      ATOM HA   TYPE=HA  CHARge=0.100  END
      ATOM CB   TYPE=CT  CHARge=-0.200 END
      ATOM HB1  TYPE=HA  CHARge=0.100  END
      ATOM HB2  TYPE=HA  CHARge=0.100  END
      ATOM CG   TYPE=CT  CHARge=-0.100 END
      ATOM HG1  TYPE=HA  CHARge=0.100  END
      ATOM HG2  TYPE=HA  CHARge=0.100  END
      ATOM HG3  TYPE=HA  CHARge=0.100  END
      ATOM C    TYPE=C   CHARge=0.480  END
      ATOM O    TYPE=O   CHARge=-0.480 END

      BOND CB CA     BOND CG CB
      BOND N HN      BOND N CA     BOND O C       BOND C CA
      BOND CA HA     BOND CB HB1   BOND CB HB2    BOND CG HG1
      BOND CG HG2    BOND CG HG3

   END

   residue bmt                              {*Define special residue BMT.*}
      ATOM N    TYPE=NH1 CHARge=-0.360 END
      ATOM HN   TYPE=H   CHARge=0.260  END
      ATOM CA   TYPE=CT  CHARge=0.000  END
      ATOM HA   TYPE=HA  CHARge=0.100  END
      ATOM CB   TYPE=CT  CHARge=0.180  END
      ATOM HB   TYPE=HA  CHARge=0.100  END
      ATOM OG1  TYPE=OH  CHARge=-0.680 END
      ATOM HG1  TYPE=H   CHARge=0.400  END
      ATOM CG2  TYPE=CT  CHARge=-0.300 END
      ATOM HG2  TYPE=HA  CHARge=0.100  END
      ATOM CD1  TYPE=CT  CHARge=-0.300 END
      atom hd11 type=ha  charge=0.1 end
      atom hd12 type=ha  charge=0.1 end
      atom hd13 type=ha  charge=0.1 end
      atom cd2 type=ct  charge=0.0 end
      atom hd21 type=ha  charge=0.0 end
      atom hd22 type=ha  charge=0.0 end
      atom ce type=ct charge=0.0 end
      atom he type=ha charge=0.0 end
      atom ch type=ct charge=0.0 end
      atom hh type=ha charge=0.0 end
      atom cp type=ct charge=-0.3 end
      atom hp1 type=ha  charge=0.1 end
      atom hp2 type=ha  charge=0.1 end
      atom hp3 type=ha  charge=0.1 end
      ATOM C    TYPE=C   CHARge=0.480  END
```

```
      ATOM O    TYPE=O   CHARge=-0.480 END

      BOND CB CA    BOND OG1 CB    BOND CG2 CB     BOND N HN
      BOND N CA     BOND O C       BOND C CA       BOND CA HA
      BOND CB HB    BOND OG1 HG1   bond cg2 hg2    bond cg2 cd1
      bond cd1 hd11 bond cd1 hd12  bond cd1 hd13   bond cg2 cd2
      bond cd2 hd21 bond cd2 hd22  bond cd2 ce     bond ce he
      bond ce  ch   bond ch  hh    bond ch cp  bond cp hp1
      bond cp hp2   bond cp hp3

      IMPROPER ch  cd2 he   ce
      IMPROPER ce  hh  cp  ch

   end
end

parameter
   @TOPPAR:parallhdg.pro                  {*Read all-h parameters, append.*}

   improper ct ct ha ct 250.0 0 0.0       {*Parameters that are needed for*}
                                          {*special residues.            *}
end

segment
   name="    "
   chain                                 {*Definition for peptide linkage.*}

      LINK PEPP    HEAD - *     TAIL + PRO     END          {*LINK to PRO.*}
      LINK PEPT    HEAD - *     TAIL + *       END

      coordinates @csa_ini.pdb
   end
end

                                                  {*Make cyclic peptide.*}
patch pept
   reference="-"=( resid 11 )
   reference="+"=( resid 1 )
end
                                                       {*Make D-ALA.*}
patch ltod
   reference=nil=( resid 8 )
end

                                                {*Make CIS-peptide bond.*}
patch CISP
   reference=-=( resid 3 )
   reference=+=( resid 4 )
end
                                                     {*Methylate groups.*}
patch m reference=nil=( resid 1 )  end
patch m reference=nil=( resid 3 )  end
patch m reference=nil=( resid 4 )  end
patch m reference=nil=( resid 6 )  end
patch m reference=nil=( resid 9 )  end
patch m reference=nil=( resid 10 ) end
patch m reference=nil=( resid 11 ) end

coordinates @csa_ini.pdb

flags exclude vdw elec end               {*Do hydrogen building w/o vdw*}
                                         {*and elec.                  *}

hbuild                                  {*This statement builds missing  *}
   selection=( hydrogen )               {*hydrogens, which are needed for*}
   phistep=45                           {*the force field.               *}
end

constraints fix=( not hydrogen ) end     {* Minimize hydrogen positions. *}
flags include vdw elec end
```

```
minimize powell
   nstep=40
end
constraints fix=( not all ) end

write coordinates output=csa_gen.pdb end          {*Write out coordinates.*}

write structure output=csa_gen.psf end   {*This writes the structure file.*}

stop
```

3.13.4 A Protein Structure with Water or Ligands

This example shows how to include information about water geometry and parameters in the molecular structure generation. The user has to be sure to use the same parameter specifications in subsequent files that need the parameters. This information can be appended directly to the appropriate parameter file.

```
remarks  file  generate/generatewater.inp
remarks  Sample: generate protein structure with waters and zinc ions

topology
   @TOPPAR:tophcsdx.pro

                                                        {*Append topology.*}

   autogenerate angles=true end

   MASS   HT     1.00800                                  {*Water hydrogen.*}
   MASS   OT    15.99940                                    {*Water oxygen.*}
   RESIdue H2O                                          {*TIPS3P water model.*}
      GROUp
         ATOM OH2  TYPE=OT   CHARge= -0.834  END
         ATOM H1   TYPE=HT   CHARge=  0.417  END
         ATOM H2   TYPE=HT   CHARge=  0.417  END
      BOND OH2  H1
      BOND OH2  H2
   END {H2O}

   MASS   ZN    65.37000                                                {*Zinc.*}
   RESIdue ZN                                    {*Zinc ion with +2 charge.*}
      GROUp
          ATOM ZN TYPE=ZN    CHARGE=2.00   END
   END {ZN}
end

parameter

  @TOPPAR:parhcsdx.pro
                                        {*Append parameters for waters.*}
  BOND HT   OT     450.0       0.9572
  ANGLE HT   OT   HT      55.0     104.52

                                        {*For solute-water interactions.*}
  NONBONDED OT        0.1591   2.8509     0.1591   2.8509
  NONBONDED HT        0.0498   1.4254     0.0498   1.4254

                                         {*For water-water interactions.*}
  !---------------A-------------B-------------A14----------B14-----
  nbfix ot  ot  581980.4948  595.0436396     581980.4948  595.0436396
  nbfix ht  ht  3.085665E-06 7.533363E-04    3.085665E-06 7.533363E-04
  nbfix ht  ot  327.8404792  10.47230620     327.8404792  10.47230620

```

```
                                                {*Append parameters for zinc.*}
  nonbonded ZN        0.1000   1.1500       0.1000 1.1500

  nbonds                                      {*This statement specifies the*}
     atom cdie shift eps=1.0  e14fac=0.4      {*nonbonded interaction energy*}
     cutnb=7.5 ctonnb=6.0 ctofnb=6.5          {*options.  Note the reduced  *}
     nbxmod=5 vswitch                         {*nonbonding cutoff to save   *}
  end                                         {*some CPU time.              *}
end

                                             {*Split the coordinate file into*}
                                             {*two files, one containing the *}
                                             {*protein coordinates, the other*}
                                             {*all the water and             *}
                                             {*zinc coordinates.             *}

segment                                            {*First, generate protein.*}

  name="    "                                  {*Should match columns 73-76 in*}
                                               {*test.pdb_pro.                 *}
  chain
    @TOPPAR:toph19.pep
    coordinates @test.pdb_protein
  end
end
coordinates @test.pdb_protein

segment                                            {*Generate water and zinc.*}

  name="    "                                  {*Should match columns 73-76 in*}
                                               {*test.pdb_water.               *}
  chain
    coordinates @test.pdb_water
  end
end
coordinates @test.pdb_water

flags exclude vdw elec end                     {*Do hydrogen building w/o vdw*}
                                               {*and elec.                     *}

hbuild                                         {*This statement builds missing*}
   selection=( hydrogen )                      {*hydrogens, which are needed  *}
   phistep=45                                  {*for the force field.         *}
end

constraints fix=( not hydrogen ) end         {* Minimize hydrogen positions. *}
flags include vdw elec end
minimize powell
   nstep=40
end
constraints fix=( not all ) end

write coordinates output=generate.pdb end

write structure output=generate.psf end
stop
```

3.13.5 A Protein Structure with a Cofactor or Substrate

The following example shows how to generate a molecular structure for a protein-ligand complex. The parameter learn statement (Section 3.4) is used to derive equilibrium (target) values for bond lenghts, bond angles, and improper angles.

```
remarks  file  generate/generateco.inp
remarks  Sample: generate protein structure with a cofactor.  Equilibrium
```

```
remarks  (target) parameters for the co-factor are learned from Cartesian
remarks  coordinates. Protein parameters are obtained from the standard
remarks  "parhcsdx.pro" parameter set.

topology
   @TOPPAR:tophcsdx.pro

   autogenerate angles=true end

                          {* Define default mass for P. The other masses *}
                          {* are already defined in file "tophcsdx.pro". *}
   MASS  P    30.97400

   RESIdue PMP                            {* Pyridoxamine phosphate; the   *}
                                          {* name PMP has to match what is *}
                                          {* used in the coordinate file.  *}
    GROUP

                     {* All PMP charges are set to 0 which is appropriate *}
                     {* for refinement purposes.  The PMP atom types are  *}
                     {* used for assigning the default mass for each      *}
                     {* atom.  The PMP atom types are unimportant for     *}
                     {* the parameterization since the parameters are     *}
                     {* learned from the PMP coordinates.                 *}

      ATOM P    TYPE=P       CHARge=0.    END
      ATOM OP1  TYPE=O       CHARge=0.    END
      ATOM OP2  TYPE=O       CHARge=0.    END
      ATOM OP3  TYPE=O       CHARge=0.    END
      ATOM OP4  TYPE=O       CHARge=0.    END
    GROUP
      ATOM C5A  TYPE=CH2E    CHARge=0.    END
      ATOM C5   TYPE=C       CHARge=0.    END
    GROUP
      ATOM N1   TYPE=NH2     CHARge=0.    END
      ATOM C6   TYPE=CH1E    CHARge=0.    END
    GROUP
      ATOM C2   TYPE=C       CHARge=0.    END
      ATOM C2A  TYPE=CH3E    CHARge=0.    END
    GROUP
      ATOM C3   TYPE=C       CHARge=0.    END
      ATOM O3   TYPE=O       CHARge=0.    END
      ATOM H3   TYPE=H       CHARge=0.    END
    GROUP
      ATOM C4   TYPE=C       CHARge=0.    END
      ATOM C4A  TYPE=CH2E    CHARge=0.    END
    GROUP
      ATOM N4   TYPE=MMM     CHARge=0.    END
      ATOM H41  TYPE=H       CHARge=0.    END
      ATOM H42  TYPE=H       CHARge=0.    END

    BOND P OP1     BOND P OP2      BOND P OP3      BOND P OP4
    BOND OP4 C5A   BOND C5A C5     BOND N1 C2      BOND N1 C6    BOND C2 C2A
    BOND C2 C3     BOND C3  O3     BOND O3 H3
    BOND C3 C4     BOND C4 C4A     BOND C4A N4     BOND N4 H41   BOND N4 H42
    BOND C4 C5     BOND C5 C6


                     {* Impropers are specified to define groups of      *}
                     {* atoms with rigid planar or tetrahedral geometry. *}
    IMPRoper N1 C2 C3 C4
    IMPRoper C2 C3 C4 C5
    IMPRoper C3 C4 C5 C6
    IMPRoper C4 C5 C6 N1
    IMPRoper C5 C6 N1 C2
    IMPRoper C6 N1 C2 C3
    IMPRoper O3 C2 C4 C3
    IMPRoper C2A N1 C3 C2
    IMPRoper C5A C4 C6 C5
    IMPRoper C4A C3 C5 C4
```

```
73      IMPRoper H42 C4A H41 N4
74
75      end
76  end
77
78                                                          {*Generate protein.*}
79  segment
80    name="    "
81    chain
82      @TOPPAR:toph19.pep
83      coordinates @../generate/maat.pdb
84    end
85  end
86                                                             {*Rename atoms. *}
87  vector do (name="O") ( name OT1 )
88  vector do (name="OT") ( name OT2 )
89  vector do (name="CD1") ( name CD and resname ile )
90
91  coordinates @../generate/maat.pdb
92
93                                                              {*Generate PMP.*}
94  segment
95    name="    "
96    chain
97      coordinates @../generate/maat.pmp
98    end
99  end
100 coordinates @../generate/maat.pmp
101
102
103
104 parameter
105   @TOPPAR:parhcsdx.pro
106
107               {* Learn parameters for PMP from the cartesian coordinates. *}
108               {* Parameters involving hydrogens have to be set explicitly *}
109               {* since the cartesian coordinates are unavailabe in this   *}
110               {* case.                                                    *}
111
112         {* Learn equilibrium parameters from known cartesian coordinates. *}
113   learn initiate sele=(resname PMP and known ) mode=nostatistics end
114   learn accumulate end
115   learn terminate end
116
117                     {* Set energy constants. We are using generic values. *}
118   BOND (resname PMP  ) (resname PMP ) 500. TOKEN
119   ANGLE (resname PMP ) (resname PMP )
120                        (resname PMP ) 500. TOKEN
121   IMPR (resname PMP )  (resname PMP )
122        (resname PMP )  (resname PMP ) 500. TOKEN TOKEN
123
124                           {* Set nonbonded parameters (epsilon and sigma) *}
125                           {* for the Lennard-Jones potential.             *}
126   NONBonded ( name C* and resname PMP ) 0.1  3.5     0.1  3.5
127   NONBonded ( name H* and resname PMP ) 0.05 1.4     0.05 1.4
128   NONBonded ( name O* and resname PMP ) 0.1  3.4     0.1  3.4
129   NONBonded ( name N* and resname PMP ) 0.1  3.4     0.1  3.4
130   NONBonded ( name P* and resname PMP ) 0.58 3.38    0.58 3.38
131
132        {* Set equilibrium parameters for hydrogens bonded to heavy atoms. *}
133        {* These parameters are needed for the hydrogen building procedure.*}
134        {* Once the hydrogens have been built, the following statement are *}
135        {* not required in subsequent jobs.                                *}
136
137   BOND (name H* and resn PMP )  (resn PMP  ) 500. 1.
138   ANGLE (name H4* and resn PMP  )  (resn PMP  ) (resn PMP ) 500. 120.
139   ANGLE (name H3  and resn PMP  )  (resn PMP  ) (resn PMP ) 500. 110.
140
141 end
142
```

```
flags exclude vdw elec impr end     {* Do quick hydrogen building without *}
                                    {* nonbonded interactions.   Impropers*}
                                    {* are excluded as well to avoid      *}
                                    {* problems with unknown parameters.  *}

hbuild                                   {* This statement builds missing *}
   selection=( hydrogen )                {* hydrogens.                    *}
   phistep=45                            {* for the force field.          *}
end

constraints fix=( not hydrogen ) end      {* Minimize hydrogen positions. *}
flags include vdw elec end
minimize powell
   nstep=40
end
constraints fix=( not all ) end

write coordinates output=generateco.pdb end

write structure output=generateco.psf end

stop

{* ------------------------------------------------------------------------ *}
{* In all subsequent jobs one has to re-create the PMP parameters           *}
{* using the generated coordinates as a reference.  All jobs must           *}
{* begin with the following statements:                                     *}


{===>} structure @generateco.psf end              {* Read structure file. *}

{===>} coor @generateco.pdb   {* Read reference coordinates for learning. *}


parameter
  @TOPPAR:parhcsdx.pro

              {* Learn parameters for PMP from the cartesian coordinates. *}

        {* Learn equilibrium parameters from known cartesian coordinates. *}
   learn initiate sele=(resname PMP and known ) mode=nostatistics end
   learn accumulate end
   learn terminate end

                     {* Set energy constants. We are using generic values. *}
   BOND (resname PMP  ) (resname PMP ) 500. TOKEN
   ANGLE (resname PMP ) (resname PMP )
                        (resname PMP ) 500. TOKEN
   IMPR (resname PMP )  (resname PMP )
        (resname PMP )  (resname PMP ) 500. TOKEN TOKEN

                           {* Set nonbonded parameters (epsilon and sigma) *}
                           {* for the Lennard-Jones potential.             *}
   NONBonded ( name C* and resname PMP ) 0.1  3.5     0.1  3.5
   NONBonded ( name H* and resname PMP ) 0.05 1.4     0.05 1.4
   NONBonded ( name O* and resname PMP ) 0.1  3.4     0.1  3.4
   NONBonded ( name N* and resname PMP ) 0.1  3.4     0.1  3.4
   NONBonded ( name P* and resname PMP ) 0.58 3.38    0.58 3.38

end

{===>} coor @generateco.pdb                  {* Read current coordinates.*}

          {* Now the parameters are well-defined and one can proceed with *}
          {* any X-PLOR task.                                             *}
```

3.13.6 A Protein Structure with a Metal Cluster

```
remarks  file  generate/generatemetal.inp
remarks  Sample: generate protein structure with metal cluster

topology
   @TOPPAR:tophcsdx.pro
                                                {*Append the metal cluster*}
   autogenerate angles=true end                 {*topology.               *}

   mass SC 32.0600
   mass FE 55.847


   RESIdue FS4                                 {*Generate the cluster itself.  *}
                                               {*The name FS4 has to match what*}
                                               {*you are using for the residue *}
    GROUP                                      {*name in the coordinate file.  *}
      ATOM FE1    type FE  charge  2.5 end
      ATOM FE2    type FE  charge  2.5 end
      ATOM FE3    type FE  charge  2.5 end
      ATOM FE4    type FE  charge  2.5 end
      ATOM S1     type SC   charge -2.0 end
      ATOM S2     type SC   charge -2.0 end
      ATOM S3     type SC   charge -2.0 end
      ATOM S4     type SC   charge -2.0 end

      bond FE1 S1  bond FE1 S2  bond FE1 S3    {*These are connectivities.*}
      bond FE2 S1  bond FE2 S2  bond FE2 S4
      bond FE3 S1  bond FE3 S3  bond FE3 S4
      bond FE4 S2  bond FE4 S3  bond FE4 S4

   END
                                             {*Generate the patch topology,*}
                                             {*which will be used in the   *}
   PREsidue PFS4                             {*PATCh command.              *}
     GROUp
       MODIfy ATOM 1CB           CHARge=0.00  END
       MODIfy ATOM 1SG  TYPE=SC  CHARge=-1.00 END
     GROUp
       MODIfy ATOM 2CB           CHARge=0.00  END
       MODIfy ATOM 2SG  TYPE=SC  CHARge=-1.00 END
     GROUp
       MODIfy ATOM 3CB           CHARge=0.00  END
       MODIfy ATOM 3SG  TYPE=SC  CHARge=-1.00 END
     GROUp
       MODIfy ATOM 4CB           CHARge=0.00  END
       MODIfy ATOM 4SG  TYPE=SC  CHARge=-1.00 END

     ADD BOND 5FE1  1SG                      {*These are the covalent bonds*}
     ADD BOND 5FE2  2SG                      {*between the CYSs and the    *}
     ADD BOND 5FE3  3SG                      {*cluster.                    *}
     ADD BOND 5FE4  4SG

                                             {*Autogenerate does not work yet*}
                                             {*for patch residues.          *}

     ADD ANGLe  1CB 1SG 5FE1          ADD ANGLe  2CB 2SG 5FE2
     ADD ANGLe  3CB 3SG 5FE3          ADD ANGLe  4CB 4SG 5FE4

     ADD ANGLe  1SG 5FE1 5S2          ADD ANGLe  1SG 5FE1 5S1
     ADD ANGLe  1SG 5FE1 5S3

     ADD ANGLe  2SG 5FE2 5S2          ADD ANGLe  2SG 5FE2 5S1
     ADD ANGLe  2SG 5FE2 5S4

     ADD ANGLe  3SG 5FE3 5S3          ADD ANGLe  3SG 5FE3 5S1
     ADD ANGLe  3SG 5FE3 5S4

```

```
      ADD ANGLe  4SG 5FE4 5S2          ADD ANGLe  4SG 5FE4 5S3
      ADD ANGLe  4SG 5FE4 5S4
 END
end

parameter

  @TOPPAR:parhcsdx.pro
                                                {*Append parameters for metal*}
  bonds SC FE   1000.0 2.33                     {*cluster.                    *}
  bonds FE FE   1000.0 2.75
  bonds SC SC   1000.0 3.60
  bonds CH2E SC 1000.0 1.81


  angle SC FE SC    500.0 112.0
  angle FE SC FE    500.0 75.0
  angle FE SC CH2E  500.0 109.5
  angle CH1E CH2E SC 500.0 114.400


!                 eps     sigma       eps(1:4) sigma(1:4)
!                 (kcal/mol) (A)
!                 ---------------------------------------
  nonbonded FE    0.1     1.15        0.1      1.15
  nonbonded SC    0.0430  3.3676      0.0430   3.3676

  nbonds                                    {*This statement specifies the*}
     atom cdie shift eps=1.0  e14fac=0.4    {*nonbonded interaction energy*}
     cutnb=7.5 ctonnb=6.0 ctofnb=6.5        {*options.  Note the reduced  *}
     nbxmod=5 vswitch                       {*nonbonding cutoff to save   *}
  end                                       {*CPU time.                   *}
end
                                          {*Split the coordinate file into*}
                                          {*two files, one containing the *}
                                          {*protein coordinates, the other*}
                                          {*the metal cluster             *}
                                          {*coordinates.                  *}

                                               {*First, generate protein.*}
segment
  name="2FXB"
  chain
    @TOPPAR:toph19.pep
    coordinates @test_2fxb_protein.pdb
  end
end
coordinates @test_2fxb_protein.pdb

                                            {*Now generate metal cluster.*}
segment
  name="2FXB"
  chain
    coordinates @test_2fxb_metal.pdb
  end
end
coordinates @test_2fxb_metal.pdb
                                             {*Now generate the          *}
                                             {*covalent links between the*}
patch  PFS4                                  {*cluster and the protein.  *}
    reference=1=( resid 11 )
    reference=2=( resid 14 )
    reference=3=( resid 17 )
    reference=4=( resid 61 )
    reference=5=( resid 82 )
end

flags exclude vdw elec end                  {*Do hydrogen building w/o vdw*}
                                            {*and elec.                   *}
hbuild                                    {*This statement builds missing  *}
```

```
138    selection=( hydrogen )                {*hydrogens, which are needed for*}
139     phistep=45                           {*the force field.              *}
140  end
141
142  constraints fix=( not hydrogen ) end      {* Minimize hydrogen positions. *}
143  flags include vdw elec end
144  minimize powell
145     nstep=40
146  end
147  constraints fix=( not all ) end
148
149  write coordinates output=generatemetal.pdb end
150
151  write structure output=generatemetal.psf end
152  stop
```

3.13.7 Oligonucleotides

The following example shows how to generate a DNA double-strand. Nucleotides are usually stored in their oxy-form in the topology file and have to be converted into the deoxy-form using an appropriate patch statement.

```
1   remarks  file  generate/generatedna.inp
2   remarks  Generate structure file for a dodecamer
3
4   topology  @TOPPAR:toph11.dna  end           {*Read topology file for dna.*}
5                                              {*This file is in subdirectory*}
6                                              {*TOPPAR.                     *}
7   parameter
8
9     @TOPPAR:param11bx.dna                    {*This file is in subdirectory*}
10                                             {*TOPPAR.                     *}
11
12    nbonds                                   {*This statement specifies the*}
13       atom cdie shift eps=1.0  e14fac=0.4   {*nonbonded interaction energy*}
14       cutnb=7.5 ctonnb=6.0 ctofnb=6.5       {*options.  Note the reduced  *}
15       nbxmod=5 vswitch                      {*nonbonding cutoff to save   *}
16    end                                      {*CPU time.                   *}
17  end
18
19                                             {*We are generating one strand*}
20                                             {*at a time.                  *}
21  segment
22    name="    "                            {*This name has to match the    *}
23                                           {*four characters in columns 73-*}
24                                           {*76 in the coordinate          *}
25                                           {*file; in XPLOR this name is   *}
26    chain                                  {*referred to as SEGId.         *}
27        LINK NUC  HEAD - *  TAIL + *  END
28        FIRST  5TER  TAIL + * END           {*5-terminus without phosphate.*}
29        LAST 3TER  HEAD - * END             {*3-terminus.                  *}
30
31        coordinates @strand1.pdb            {*Interpret coordinate file to*}
32    end                                     {*obtain sequence.             *}
33  end
34
35                                            {*It comes as RNA; now we have*}
36                                            {*to apply deoxy patches for  *}
37                                            {*each residue.               *}
38
39 for $1 in ( 402 403 404 405 406 407 408 409 410 411
40                      412 413 414 415 416 417 418 419) loop main
41     patch deox reference=nil=( resid $1 ) end
42  end loop main
43
44
45  coordinates @strand1.pdb                     {*Here we actually read the*}
```

```
                                                  {*coordinates.            *}

{-------------------------------------------------------------------------}
 segment                                            {*Generate second strand.*}

   name="    "
   chain
       LINK NUC  HEAD - *  TAIL + *  END
       FIRST  5TER  TAIL + * END
       LAST 3TER  HEAD - * END
       coordinates @strand2.pdb
   end
 end

 for $1 in ( 421 422 423 424 425 426 427 428 429 430
                   431 432 433 434 435 436 437 438) loop main
    patch deox reference=nil=( resid $1 ) end
 end loop main


 coordinates @strand2.pdb


{-------------------------------------------------------------------------}

 flags exclude vdw elec end                  {*Do hydrogen building w/o vdw*}
                                             {*and elec.                   *}

 hbuild                                     {*This statement builds missing*}
    selection=( hydrogen )                  {*hydrogens, which are needed  *}
    phistep=45                              {*for the force field.         *}
 end

 constraints fix=( not hydrogen ) end      {* Minimize hydrogen positions. *}
 flags include vdw elec end
 minimize powell
    nstep=40
 end
 constraints fix=( not all ) end

 write coordinates output=generatedna.pdb end      {*Write out coordinates.*}

 write structure output=generatedna.psf end     {*Write the structure file.*}

          {* Oligonucleotides usually require additional restraints during *}
          {* refinement.  The following file must be included in all       *}
          {* subsequent files.                                            *}

 @brestraints.inp

 stop
```

Sugar puckers, hydrogen bonding, and base planarity often need to be restrained for X-ray or NMR structure determination or refinement. This is not a problem with X-PLOR or the energy function used but rather a consequence of an unfavorable observable to parameter ratio. To get around these problems one can restrain the base planarity and the sugar puckers using the restraints planarity statement (Section 7.3) and restraints dihedral statement (Section 7.2), respectively. Hydrogen bond restraints can be introduced by using the distance restraints (Section 18.1).

The following example file illustrates the use of these restraints for B-form DNA. This file must be included in all structure determination or refinement protocols of the oliognucleotide. Attention must be paid to the inclusion of

the additional energy flags (noe, plan, cdih). Please note that in this example an existing database of distance or dihedral angle restraints is erased.

```
 remarks  file  generate/brestraints.inp
 remarks  Creates restraints for B-form DNA
 remarks  This file needs to be included after the structure file has been
 remarks  read.
 remarks  Please note that this example file overwrites the NOE and
 remarks  RESTraints DIHEdral databases.

set message=off end
set echo=off end

{-------------------------------------------------------------------------}
                                      {* Apply base planarity restraints. *}
restraints plane

  for $1 in (

{====>}                              {* Select nucleotides to be restrained. *}
   402 438    403 437    404 436     405 435     406 434     407 433
   408 432    409 431    410 430     411 429     412 428     413 427
   414 426    415 425    416 424     417 423     418 422     419 421
             ) loop plan
      group
    selection=(
    (not( hydro or name P or name O1P or name O2P or name O5' or name H5T or
  name H1' or name H2' or name H3' or name H4' or name H5'
                  or name H2"" or name H5"" or
  name O3' or name C2' or name C3'
                  or name C4' or name C5' or name O4' or name H3T)
                  and resid $1 ))
    weight=400.0
      end

  end loop plan

end

flags include plan end

{-------------------------------------------------------------------------}
                               {* Apply sugar pucker restraints for B-DNA. *}

restraints dihedral
           nassign=600

  for $1 in (

{====>}                              {* Select nucleotides to be restrained. *}
   402 438    403 437    404 436     405 435     406 434     407 433
   408 432    409 431    410 430     411 429     412 428     413 427
   414 426    415 425    416 424     417 423     418 422     419 421

            ) loop dihe

           assign  ( resid $1 and name c1' )
                   ( resid $1 and name c2' )
                   ( resid $1 and name c3' )
                   ( resid $1 and name c4' ) 20.0 -34.9 0.0 2

           assign  ( resid $1 and name c5' )
                   ( resid $1 and name c4' )
                   ( resid $1 and name c3' )
                   ( resid $1 and name c2' ) 20.0 -86.4 0.0 2

           assign  ( resid $1 and name c1' )
                   ( resid $1 and name o4' )
                   ( resid $1 and name c4' )
```

```
                    ( resid $1 and name c5' ) 20.0 106.4 0.0 2
  end loop dihe

  scale=20.0

end


{-------------------------------------------------------------------------}
                  {* Hydrogen bond restraints for Watson-Crick base pairs. *}

 noe
   nres=3000
   class h
   ceiling=1000
   averaging h cent
   potential h square
   sqconstant h 1.
   sqexponent h 2
   scale h 50.
 end

evaluate ($counter=0)
for $1 in (

{====>}                              {* Select base pairs to be restrained. *}
   402 438    403 437    404 436     405 435     406 434     407 433
   408 432    409 431    410 430     411 429     412 428     413 427
   414 426    415 425    416 424     417 423     418 422     419 421

            ) loop noe

   evaluate ($counter=$counter+1)
   if     ($counter=1) then  evaluate($a=$1)
   elseif ($counter=2) then
      evaluate($b=$1)
      evaluate ($counter=0)

      vector show elem ( resn ) ( resid $a and name C1' )
      evaluate ($aa=$result)
      vector show elem ( resn ) ( resid $b and name C1' )
      evaluate ($bb=$result)

      noe
       if ($aa=THY) then                                  {* T:A basepair. *}
         assign (resid $a and name o4) (resid $b and name n6) 2.95 0.01 0.01
         assign (resid $a and name n3) (resid $b and name n1) 2.82 0.01 0.01
         assign (resid $a and name o4) (resid $b and name n1) 3.63 0.01 0.01
         assign (resid $a and name o2) (resid $b and name n6) 5.40 0.01 0.01

       elseif ($aa=ADE) then                              {* A:T basepair. *}
         assign (resid $b and name o4) (resid $a and name n6) 2.95 0.01 0.01
         assign (resid $b and name n3) (resid $a and name n1) 2.82 0.01 0.01
         assign (resid $b and name o4) (resid $a and name n1) 3.63 0.01 0.01
         assign (resid $b and name o2) (resid $a and name n6) 5.40 0.01 0.01

       elseif ($aa=CYT) then                              {* C:G basepair. *}
         assign (resid $a and name n4) (resid $b and name o6) 2.91 0.01 0.01
         assign (resid $a and name n3) (resid $b and name n1) 2.95 0.01 0.01
         assign (resid $a and name o2) (resid $b and name n2) 2.86 0.01 0.01
         assign (resid $a and name n3) (resid $b and name n2) 3.65 0.01 0.01
         assign (resid $a and name o2) (resid $b and name o6) 5.42 0.01 0.01

       elseif ($aa=GUA) then                              {* G:C basepair. *}
         assign (resid $b and name n4) (resid $a and name o6) 2.91 0.01 0.01
         assign (resid $b and name n3) (resid $a and name n1) 2.95 0.01 0.01
         assign (resid $b and name o2) (resid $a and name n2) 2.86 0.01 0.01
         assign (resid $b and name n3) (resid $a and name n2) 3.65 0.01 0.01
         assign (resid $b and name o2) (resid $a and name o6) 5.42 0.01 0.01
       end if
```

```
137       end
138
139    end if
140 end loop noe
141
142 set message=on end
143 set echo=on end
144
145 noe
146    ceiling 1000.
147    scale * 100.
148    potential * squarewell
149 end
150
151 flags include plan cdih noe end
152
```

A similar file is available for A-DNA ("/generate/arestraints.inp").

To find out the target values for the conformation of a particular oligonucleotide coordinate set, the pick dihedral and pick bond statements (Section 5.1.2) can be used to obtain the dihedral angles and the hydrogen bond distances, respectively.

3.13.8 Virus Structures or Structures with Many Identical Units

Molecular structures with many identical units can be generated using the principles discussed above. However, it would be inefficient to generate the whole multimer in one generation protocol since the segment statement uses a pairwise atom check. Instead, a molecular structure file for a protomer should be generated. Then, the duplicate statement should be used to create the multimer. Assume that the "protomer.psf" file contains the molecular structure of a protomer with blank segment name.

If a multimer coordinate file exists, one has to split the coordinates of the multimer into several files, one for each protomer with disjoint segment names. The following shows an example for a trimeric structure (ignore the "multiple coordinates" error messages).

```
1 structure @protomer.psf
2 vector do (segid="a") ( segid " " )
3 coordinates @a.pdb                                    {* Read protomer a.*}
4 duplicate segid="b" selection=( segid "a" ) end
5 coordinates @b.pdb                                    {* Read protomer b.*}
6 duplicate segid="c" selection=( segid "a" ) end
7 coordinates @c.pdb                                    {* Read protomer c.*}
```

Note that if using the NCS strict statement (Section 16.2.1), one has to create only one protomer structure.

If a protomer coordinate file exists and one knows the appropriate transformations between the monomers, one should read the monomer, use the duplicate statement to duplicate the molecular structure as well as the coordinates, and apply appropriate coordinate transformations (marked by ...in the example below) to produce the other monomers.

```
structure @protomer.psf
coordinate @protomer.pdb
vector do (segid="a") ( segid " " )
duplicate segid="b" selection=( segid "a" ) end
coordinates rotate ... selection=( segid "b" end
coordinates translate ... selection=( segid "b" )  end
duplicate segid="c" selection=( segid "a" ) end
coordinates rotate ... selection=( segid "c" end
coordinates translate ... selection=( segid "c" )  end
```

4 Energy Function

The XPLOR energy function can be grouped into two different classes:

$$E_{TOTAL} = E_{EMPIRICAL} + E_{EFFECTIVE} \tag{4.1}$$

$E_{EMPIRICAL}$ describes the energy of the molecule(s) through an empirical energy function described below. $E_{EFFECTIVE}$ comprises restraining energy terms that use experimental information or other information:

$$\begin{aligned} E_{EFFECTIVE} \quad = \quad & E_{XREF} + E_{NOE} + E_{HARM} + \\ & E_{CDIH} + E_{NCS} + E_{DG} + E_{RELA} + E_{PLAN} \end{aligned} \tag{4.2}$$

The E_{XREF}, E_{NOE}, E_{HARM}, E_{CDIH}, E_{NCS}, E_{DG}, E_{RELA}, and E_{PLAN} terms are described in Chapters 12, 18, 7.1, 7.2, 16, 19, 21, and 7.3, respectively.

4.1 Empirical Energy Functions

Empirical energy functions describe the energy of the molecule as a function of the atomic coordinates (Lifson and Stern 1982; Burkert and Allinger 1982; Brooks et al. 1983; Némethy, Pottie, and Scheraga 1983; Weiner et al. 1984; Karplus and Petsko 1990). Most of today's empirical energy functions contain conformational and nonbonded interaction energy terms involving sets of two, three, and four atoms. A harmonic approximation is used to account for deformations in bond length and angles. Four-atom terms are used for torsion potentials. Two-atom terms are employed for the nonbonded interactions. A variety of specific parameterizations for empirical energy functions are available in X-PLOR (Section 3.6).

XPLOR's empirical energy function has the general form

$$\begin{aligned} E_{EMPIRICAL} \quad = \sum_{p=1}^{N} [\quad & w^p_{BOND} E_{BOND} + w^p_{ANGL} E_{ANGL} + \\ & w^p_{DIHE} E_{DIHE} + w^p_{IMPR} E_{IMPR} + \\ & w^p_{VDW} E_{VDW} + w^p_{ELEC} E_{ELEC} + \\ & w^p_{PVDW} E_{PVDW} + w^p_{PELE} E_{PELE} + \\ & w^p_{HBON} E_{HBON}]. \end{aligned} \tag{4.3}$$

The sum is carried out over all double selections of atoms (see Section 4.7) with weights w_n^p. The default for the double selections is one double selection involving all atoms with unity weights. The first four terms in Eq. 4.3 are conformational energy terms.

The remaining five terms in Eq. 4.3 describe nonbonded interactions. The term E_{VDW} describes the non-symmetry-related van der Waals energy, E_{ELEC} describes the non-symmetry-related electrostatic energy, E_{PVDW} describes the van der Waals energy between symmetry-related atoms, and E_{PELE} describes the symmetry-related electrostatic energy. The term E_{HBON} describes an explicit hydrogen-bonding energy. This term is used only in older parameter files.

In the next sections, the empirical energy terms are described in more detail.

4.2 Conformational Energy Terms

The term

$$E_{BOND} = \sum_{bonds} k_b(r - r_0)^2 \tag{4.4}$$

describes the covalent bond energy where the sum is carried out over all covalent bonds in the molecular structure selected by the constraints interaction statement; r is the actual bond length, k_b is energy constants, and r_0 is equilibrium constants specified by parameter statements (Section 3.2.1).

The term

$$E_{ANGL} = \sum_{angles} (k_\theta(\theta - \theta_0)^2 + k_{ub}(r_{13} - r_{ub})^2) \tag{4.5}$$

describes the bond angle energy where the sum is carried out over all bond angles in the molecular structure selected by the constraints interaction statement; k_θ and k_{ub} are energy constants and θ_0 and r_{ub} are equilibrium constants specified by parameter statements (Section 3.2.1). θ is the actual value of the angle, and r_{13} is the distance between the first and the third atom defining the angle. The second term in Eq. 4.5 is the Urey-Bradley term, which is used by certain force fields (Burkert and Allinger 1982). The default value for k_{ub} is zero.

The angle between two planes, the first being defined through atoms i,j,k and the second through atoms j,k,l, is defined as a torsion angle where the atoms i,j,k,l are specified by the dihedral and improper statements (Section 3.1.1). The terms

(4.6)

$$E_{DIHE} = \sum_{dihedrals} \sum_{i=1,m} \begin{cases} k_{\phi_i}(1 + \cos(n\phi_i + \delta_i)) \text{ if } n_i > 0 \\ k_{\phi_i}(\phi_i - \delta_i)^2 \text{ if } n_i = 0 \end{cases}$$

$$E_{IMPR} = \sum_{impropers} \sum_{i=1,m} \begin{cases} k_{\phi_i}(1+\cos(n\phi_i+\delta_i)) \text{ if } n_i > 0 \\ k_{\phi_i}(\phi_i-\delta_i)^2 \text{ if } n_i = 0 \end{cases}$$

describe the dihedral and improper energy terms. The sums are carried out over all dihedral or improper angles in the molecular structure selected by the constraints interaction statement; ϕ_i is the actual torsion angle, k_{ϕ_i} are energy constants, n_i are periodicities, m_i are multiplicities, and δ_i are phase shifts (Section 3.2.1). Note that the definition of dihedral and improper angles is identical. However, X-PLOR maintains two separate topology and parameter lists for dihedral and improper angles. Historically, improper angles are mostly used with $n = 0$ to maintain planarity or chirality, whereas dihedral angles are used with $n > 0$ to describe multi-minimum torsion potentials.

The specification of multiple dihedral or torsion angles with $m > 1$ allows one to carry out a cosine expansion of a torsion potential for a particular instance involving four atoms or four atom types. Multiple dihedral angles and improper angles have to be indicated by using the MULTiple option both in the definition of the molecule's topology (Section 3.1.1) and in the specification of the corresponding parameter (Section 3.2.1). Internally, X-PLOR stores multiple dihedral or improper angles as multiple instances of the same combination of atoms or atom types.

4.3 Nonbonded Energy Terms

The options for the van der Waals and electrostatic energy terms are selected by an appropriate nbonds statement (see Section 3.2.1). Four combinations of nbonds options are possible. The first is TRUNcation in combination with CDIE , which uses Lennard-Jones and Coulomb functions. The second involves a switched van der Waals (VSWItch) and a shifted electrostatic function (SHIFt or SHFOrce) in combination with CDIE. The third uses a switched van der Waals function (VSWItch) in combination with a switched electrostatic function (SWITch) and a $1/R$ dielectric function (RDIE) . The fourth (REPEl > 0) uses the so-called repel function, which is a purely repulsive function without attractive or electrostatic components. Any other combinations of the SHIFt, SWITch, VSHIft, VSWItch, CDIE, RDIE, TRUNCation and REPEl options are not possible and will produce an error message.

All nonbonded energy terms are truncated for atom pairs that are too close to each other (INHIbit option in the nonbonded statement, Section 3.2.1).

$$R \rightarrow \max(R, R_{inhibit}) \tag{4.7}$$

This feature reduces numerical instabilities in the case of strained initial coodinates or close contacts during high simulation temperatures.

4.3.1 Van der Waals Function

The van der Waals function is given by

$$f_{VDW}(R) = \begin{cases} \frac{A}{R^{12}} - \frac{B}{R^6} = 4\varepsilon((\frac{\sigma}{R})^{12} - (\frac{\sigma}{R})^6)\mathrm{H}(R - R_{cut}) & \text{truncation} \\ \left(\frac{A}{R^{12}} - \frac{B}{R^6}\right) SW(R, R_{on}, R_{off}) & \text{switched} \\ C_{rep}(max(0, (k^{rep}R_{min})^{irexp} - R^{irexp}))^{rexp} & \text{repel} \end{cases} \tag{4.8}$$

where H is the step function and SW is a switching function. SW has the form

$$SW(R, R_{on}, R_{off}) = \begin{cases} 0 & \text{if } R > R_{off} \\ \frac{(R_{off}^2 - R^2)^2 * (R_{off}^2 - R^2 - 3(R_{on}^2 - R^2))}{(R_{off}^2 - R_{on}^2)^3} & \text{if } R_{off} > R > R_{on} \\ 1 & \text{if } R < R_{on} \end{cases} \tag{4.9}$$

For both the truncated and the switched option, the van der Waals function is described by a Lennard-Jones potential. In this potential, the attractive force is proportional to R^{-6}, while the repulsive force varies as R^{-12}. A,B and ε,σ are related to the well depth E_{min} and the minimum distance R_{min} (van der Waals radius) by

$$R_{min} = \sigma\sqrt[6]{2} \tag{4.10}$$

$$E_{min} = -\varepsilon \tag{4.11}$$

$$A = 4\sigma^{12}\varepsilon \tag{4.12}$$

$$B = 4\sigma^{6}\varepsilon \tag{4.13}$$

The repel option uses a simple repulsive potential. It is used primarily for structure refinements with X-ray crystallographic and solution NMR spectroscopic data.

The NBON statement (Section 3.2.1) defines ε, σ for the Lennard-Jones potential between identical atom types. Between different atom types, the following combination rule is used by default:

$$\sigma_{ij} = \frac{\sigma_{ii} + \sigma_{jj}}{2} \tag{4.14}$$

$$\varepsilon_{ij} = \sqrt{\varepsilon_{ii}\varepsilon_{jj}} \tag{4.15}$$

The NBFix statement allows one to deviate from this combination rule; i.e, one can explicitly specify the A, B coefficients for an atom type pair (see Section 3.2.1).

Minimization of the empirical potential energy for initial coordinates with very close nonbonded contacts is often ill behaved because the Lennard-Jones potential produces a very large gradient reflecting the close contacts. To avoid this problem, the Lennard-Jones and electrostatic potential can be replaced with the repel potential, which is softer and purely repulsive.

4.3.2 Electrostatic Function

The electrostatic function is given by

(4.16)

$$f_{ELEC}(R) = \begin{cases} Q_iQ_j\frac{C}{\varepsilon_o R}\text{heavy}(R - R_{cut}) & \text{for pure truncation} \\ Q_iQ_j\frac{C}{\varepsilon_o R}(1 - \frac{R^2}{R_{off}^2})^2 & \text{for potential shifting option} \\ Q_iQ_j\frac{C}{\varepsilon_o R}(\frac{1}{R} + \frac{R^b}{bR_{off}^{b+1}} + \frac{b+1}{bR_{off}}) & \text{for force shifting option} \\ Q_iQ_j\frac{C}{\varepsilon_o R^2}\ SW(R, R_{on}, R_{off}) & \text{for 1/R option} \\ 0 & \text{for repel option} \end{cases}$$

The electrostatic function is computed using the atomic charges provided by the CHARge specification in the atom statement (see Section 3.1.1). ε_0 is the dielectric constant, which can be defined by the EPS statement; r_{on} and r_{off} are defined by the statements CTONNB and CTOFNB respectively (see Section 3.2.1). Because of the need to limit the number of pair interactions and to avoid discontinuities in the forces (to conserve energy during dynamics), several schemes for truncating the electrostatic potential are used. The $1/R$ option introduces an approximate solvent screening term in the dielectric constant by setting the constant equal to $\varepsilon_0 R$. The $1/R$ dielectric option was originally developed because the execution of square roots was expensive on certain obsolete computers. There is no physical justification for the 1/R dielectric, but it is still in widespread use, in particular for simulations in vacuum. The shifted options modifie the radial function so that the energy and forces go to zero at the cutoff distance. Potential shifting results in a smooth decay of the energy to zero while force shifting results in a smooth decay of the forces to zero (for further reading see Steinbach and Brooks, 1994). Force shifting with b=1 should preferably be used, in combination with a long cutoff (CTOFNB=12 Å), when studying the dynamics of a biomolecule in water.

4.3.3 Intramolecular Interactions

The intramolecular interaction energy is the summation of the individual nonbonded interaction energies for pairs of atoms within the current molecular structure, e.g., a single molecule or a crystallographic asymmetric unit.

$$E_{ELEC} = \sum_{i<j} f_{ELEC}(R_{ij}) + e_{14} \sum_{(i,j)\in\{1-4\}} f_{ELEC}(R_{ij}) \tag{4.17}$$

$$E_{VDW} = \sum_{i<j} f_{VDW}(R_{ij}) + \sum_{(i,j)\in\{1-4\}} f_{VDW}(R_{ij}) \tag{4.18}$$

The summation extends over all pairs of atoms ($i < j$) that satisfy the cutoff criteria specified by CUTNb and ATOM or GROUP (nbonds statement, see Section 3.2.1) and that are selected by the constraints interaction

statement. The program's interpretation of the cutoff points depends on the type of cutoff used. For the group cutoff option, the program first executes a group-by-group centroid search to determine which centroids are within the particular cutoff value CUTNb of each other. Then the program calculates the energy using every atom in both groups. For the atom cutoff, the nonbonded interactions are included on an atom-by-atom basis. However, the pairwise search is computationally expensive. A great reduction in computational time is achieved by first searching for nearest neighbors among the nonbonded groups and then searching for atom pairs between selected group pairs.

The computational time is further reduced by introducing an approximation, storing the atomic pair indices ($i < j$) that satisfy $R_{ij} \leq R_{cut}$ in a list that is updated only when any atom has moved more than the amount specified by TOLErance. For both switched and shifted nonbonded options (see Section 3.2.1), the distance R_{ij} at which the energy becomes zero is given by R_{off}. Thus, the nonbonded energy calculations become independent of the update frequencies if CUTNB $\geq$ CTOFNB + 2TOLErance.

There are a number of cases for nonbonded interactions that must not be computed, e.g., interactions between covalently bonded atoms. Covalently bonded exclusions are automatically generated by X-PLOR using information about atom connectivity. In addition, certain exclusions can be added manually by the EXCLude statement, which is an atom statement (see Section 3.1.1). The NBXMod statement (see Section 3.2.1) has several options for automatically excluding 1–2, 1–2 and 1–3, and 1–2, 1–3, and 1–4 interactions in the molecule. In the case of NBXMod=$\pm$5, the 1–4 interactions are treated in a special way. The electrostatic 1–4 interactions are scaled by e_{14}, and the van der Waals interactions use a special 1–4 set of parameters for ε, θ, A and B. In the case of NBXMod#$\pm$5, 1–4 interactions are treated as normal nonbonded interactions, and the second terms of the right-hand side of Eqs. 4.17 and 4.18 become zero.

4.3.4 Crystallographic Symmetry Interactions

Crystallographic symmetry interactions (e.g., packing interactions) between the molecule(s) located in the asymmetric unit and all symmetry-related molecules surrounding the asymmetric unit are computed by

$$E_{PVDW} = \sum_{S=1}^{n_S} \sum_{i<j} f_{VDW}(R_{iSj}) \tag{4.19}$$

$$E_{PELE} = \sum_{S=1}^{n_S} \sum_{i<j} f_{ELEC}(R_{iSj}) \tag{4.20}$$

where the first sum extends over all cystallographic symmetry operators $(\mathcal{O}_s, \vec{t}_s)$ and R_{iSj} is defined by

$$R_{iSj} = |\mathcal{F}^{-1} \cdot MinG(\mathcal{F} \cdot \vec{r}_i - \mathcal{O}_s \cdot \mathcal{F} \cdot \vec{r}_j + \vec{t}_s)| \tag{4.21}$$

$\mathcal{F}$ is the matrix that converts orthogonal coordinates into fractional coordinates, and the second sum extends over all pairs of atoms (i, j) for which $R_i s_j$ is less then a specified cutoff selected by the constraints interaction statement r_{cut} and r_i and r_j are the coordinates. The function $MinG(\vec{r})$ defines the minimum image distance in fractional coordinate space. It operates separately on each component of the three-dimensional vector $\vec{r}$, where the operation on each component x is given by

$$MinG(x) = sign(-x)\ int(|x| + 1/2) + x \tag{4.22}$$

The function $int(x)$ is defined as the integer part of x, and $sign(x)$ is defined as the sign of x. The nonbonding interaction energy between two atoms is independent of whether it is an intermolecular or an intramolecular interaction. The atom/group search and the update of the interaction list are carried out analogously to the intramolecular case. (Refer to Section 12.3 for the definition of symmetry operators and the unit-cell constants that define $\mathcal{F}$.)

4.3.5 Non-crystallographic Symmetry Interactions

If strict non-crystallographic symmetry is imposed, the crystal symmetry interactions cannot be computed because of nonequivalent environments (see Chapter 16). Instead, only the interaction between non-crystallographically related molecules is computed using the f_{VDW} and f_{ELEC} functional forms, i.e.,

$$E_{PVDW} = \sum_{S \in NCS} \sum_{i<j} f_{VDW}(r_i - Sr_j) \tag{4.23}$$

$$E_{PELE} = \sum_{S \in NCS} \sum_{i<j} f_{ELEC}(r_i - Sr_j) \tag{4.24}$$

The atom/group search and the update of the interaction list are carried out analogously to the intramolecular case. (Refer to Chapter 16 for the definition of non-crystallographic symmetry operators.)

4.4 The Explicit Hydrogen-Bond Term

The explicit hydrogen-bond energy term E_{HBON} is used in certain parameter sets, in particular the ones that deal with nucleic acids. In the more recent protein parameter files, this term is zero, and the effect of hydrogen bonds is taken implicitly into account by appropriate parameterization of the partial charges and van der Waals parameters. The explicit hydrogen-bond term has the form

$$E_{HBON} = \sum_{i<j} \left(\frac{A}{r_{AD}^{i}} - \frac{B}{r_{AD}^{j}}\right) \tag{4.25}$$

$$cos^m(\theta_{A-H-D})cos^n(\theta_{AA-A-H})SW(r^2_{AD}, r^2_{h_{on}}, r^2_{h_{off}})$$
$$SW[cos^2(\theta_{A-D-H}), cos^2(\theta_{h_{on}}), cos^2(\theta_{h_{off}})]$$

involving atoms AA, A, H, D (acceptor antecedent, acceptor, hydrogen, and donor heavy atom), where i, j, m, and n are positive integers and SW is the switching function defined in Eq. 4.9.

The switching function SW maintains a continuous and smooth function to the cutoff point so that derivatives are well defined at the cutoff point. The sum in Eq. 4.25 extends over all pairs of donors and acceptors (DONOr and ACCEptor statements; see Section 3.1.1) for which the donor-acceptor distance and the donor-hydrogen-acceptor angle satisfy the criteria specified by DCUT and ACUT (see Section 3.2.1). AEXP assigns the j exponent, REXP assigns the i exponent, AHEX assigns the m exponent, and AAEX assigns the n exponent; A and B are defined by the HBON statement (see Section 3.2.1). The remaining switching parameters are provided by the hbonds statement. The computational time for searching for hydrogen bonds that satisfy the cutoff criteria is reduced by introducing an approximation, storing the atomic pair indices ($i < j$) that satisfy the cutoff criteria in a list that is updated only when any atom has moved more than the value specified by TOLErance.

4.5 Turning Energy Terms On or Off

The flag statement allows the user to turn energy terms on and off. This implies that the next time the energy is calculated in X-PLOR (e.g., with the minimize Powell statement, the dynamics Verlet statement, or the energy statement), only the selected energy terms will affect the total energy and the atomic forces.

4.5.1 Syntax

FLAGs { <**flag-statement**> } **END** is invoked from the main level of X-PLOR. Note: certain energy terms that are initially off will be turned on automatically when statements related to this energy are invoked, but this is generally not the case.

<**flag-statement**>:==

EXCLude {<***energy-term***>} excludes specified energy-terms.

INCLude {<***energy-term***>} includes specified energy-terms.

<**energy-term**>:==

ANGL specifies bond angle energy (default: on).

BOND specifies covalent bond energy (default: on).

CDIH specifies dihedral angle restraints energy (default: off).

DG is a distance geometry restraint term (see Chapter 19) (default: off).

DIHE specifies dihedral angle energy (default: on).

ELEC specifies intramolecular electrostatic energy (default: on).

HARM specifies a harmonic energy that restrains the positions of the molecule (default: off).

HBON specifies explicit hydrogen-bond energy (default: off).

IMPR specifies improper dihedral angle (e.g., chirality and planarity) energy (default: on).

NCS specifies non-crystallographic positional restraint energy (default: off).

NOE specifies distance restraints (see Chapter 18) (default: off).

PELE specifies symmetry-related electrostatic energy (default: off).

PLAN specifies planarity restraints energy (default: off).

PVDW specifies symmetry-related van der Waals energy (default: off).

RELA is a complete matrix relaxation effective energy term (see Chapter 21) (default: off).

VDW specifies intramolecular van der Waals energy (default: on).

XREF specifies crystallographic effective energy (default: off).

4.5.2 Requirements

The statement can be issued at any place in the input. No check is done to determine whether the energy terms that are turned on are actually well defined.

4.5.3 Example: Turn On Energy Terms for X-ray Refinement

The following example shows how to include the effective energy representing the crystallographic residual and the symmetry-related nonbonded interactions in the energy calculation:

```
1 FLAGs
2    INCLude XREF  PVDW  PELE
3 END
```

The next example turns off all energy terms except for the covalent bond energy term:

```
1 FLAGs
2    EXCLude * INCLude BOND
3 END
```

4.6 Energy Statement

The energy statement performs a single calculation of all energy terms that are turned on. The atomic forces are also computed and stored in arrays DX, DY, and DZ (see vector-statement, Section 2.16).

Each time this statement is issued, it performs an accumulation of the partial energy terms. Average and rms values of the accumulated energy terms can also be printed by using this statement.

Upon completion of the energy calculation, symbols are declared that contain the computed energy terms. The name of the symbols is given by $<energy-term> (see Section 4.5). The overall energy (Eq. 4.1) is stored in the symbol $ENER; the rms gradient is stored in $GRAD. The value of the second energy function (Eq. 4.26) is returned in the symbol $PERT.

4.6.1 Syntax

ENERgy { <**energy-statement**> } **END** is invoked from the main level of X-PLOR. The energy calculation is carried out as soon as the END statement is given.

<**energy-statement**>:==

ACCUmulate=ENERgy|PRINt|RESEt specifies options for computing energy statistics. If ENERgy is specified, it computes, prints, and accumulates the energy. If RESEt is selected, the internal energy accumulator is reset without performing an energy calculation. If PRINt is selected, the average and rms values of the partial energy terms will be printed without an energy calculation (default: ENERgy).

4.6.2 Requirements

The requirements for the partial energy terms that are turned on have to be satisfied in order for this statement to be successful. In most cases this means that at least the molecular structure, parameters, and atomic coordinates have to be well defined.

4.6.3 Examples

The following example performs a single energy calculation:

```
1 energy end
```

The next example shows how to acquire accumulative information about the partial energy terms:

```
1 energy accumulate=reset end
2 ...
3 energy end
4 ...
```

```
energy end
...
energy accumulate=print end
```

4.7 Energy Calculation between Selected Atoms

The constraints interaction statement tells the empirical energy routines of X-PLOR to compute the energy between only two selected sets of atoms. It is convenient to speak of these two selected sets as a double selection. For two-point energy terms (such as covalent bonds and nonbonded interactions), the energy is computed if one atom of the bond belongs to the first selection and the other atom belongs to the second selection. For three-point terms (such as angles) and four-point terms (such as dihedrals), the energy is computed if at least one atom belongs to the first selection, at least one other atom belongs to the second selection, and all atoms of the three-point or four-point term belong to at least one selection. The terms that are affected are BOND, ANGL, DIHE, IMPR, VDW, ELEC, HBON, PVDW, and PELE (see Eq. 4.3). Other energy terms (e.g., X-ray or NOE terms) are not affected. The statement can be issued several times, thereby defining several double selections. In that case, the total energy and the total forces are obtained by summing over the different double selections. In addition, when a double selection is defined, the user may weight its contribution to the energy and forces by attributing a weight to each individual energy term (bonds, angles, etc.).

The constraints interaction statement also permits one to build up a second energy function, V_{total}, which is useful for purposes of analysis. To do this, a second weight, v_{pk}, is attributed to each individual energy term in each double selection. This second weighting scheme does not affect the forces or the dynamics of the system, but is purely a tool for analysis. The second energy function, V_{total}, thus has the form

$$V_{TOTAL} = \sum_{p=1}^{N} [\sum_{k} w_k^p E_k] \tag{4.26}$$

The first sum is over the N double selections ; the second is over the empirical energy terms: BOND, ANGL, DIHE, IMPR, VDW, ELEC, HBON, PVDW, and PELE. This second energy function will appear as the "PERT" term in the output and the energy symbols.

The invocation of the constraints statement will automatically erase all previous double selections. To specify multiple double selections one has to include them within the same constraints statement, e.g.,

```
constraints
  interaction=( segid "A" ) ( segid "A" )
```

```
3   interaction=( segid "B" ) ( segid "B" )
4 end
```

4.7.1 Syntax

CONStraints { < **constraints-interaction-statement** > } **END** is invoked from the main level of X-PLOR.

<**constraints-interaction-statement**>:==

INTEraction=<**selection**> <**selection**> [{ <**weight-statement**> }] specifies the two selected sets of atoms. The double selection is active until a new **CONStraints** < **constraints-interaction-statement** > statement is issued. The default double selection is a single double selection involving all atoms of the molecular structure.

<**weight-statement**>:==

VWEIghts {<***energy-term***> <**real**> } **END** applies the weight(real) to the specified energy term for computing V_{total} (default : all terms having zero VWEIghts; i.e., no perturbation analysis is being done). This Hamiltonian does not contribute to the forces, but it can be used for purposes of analysis.

WEIGhts {<***energy-term***> <**real**> } **END** applies the weight (real) to the specified energy term for the Hamiltonian E_{total} (default: all active terms having unit WEIGhts).

4.7.2 Requirements

The statement can be issued at any place in the input, after the molecular structure has been well defined in order to make the selection (see Section 2.15) work properly. The atom selections for the double selections are fragile (Section 2.15).

4.7.3 Example: Interchain Interaction Energy

The following example shows how to restrict the energy calculation to the interaction energy between two chains with segment identifiers "a" and "b." Only the energies and forces between the selected atoms are computed.

```
1 CONStraints INTEraction
2     ( segid a ) ( segid b )
3 END
```

This example does not include the intrachain energy. To include the intrachain energy, one should specify the following:

```
1 CONStraints INTEraction
2     ( segid a or segid b ) ( segid a or segid b )
3 END
```

The following example modifies the weight of the inter- and intrachain interactions in the Hamiltonian, applying a weight of 0.5 to the bond energies and forces and a factor of 0.75 to the other energy terms :

```
CONStraints INTEraction
   ( segid a or segid b ) ( segid a or segid b ) WEIGhts * 0.75 bonds 0.5 END
END
```

This example includes the intrasegment bond energy and the overall intersegment energy but excludes the intrasegment angle and dihedral, improper, and nonbonded terms from the Hamiltonian :

```
CONStraints
    INTEraction ( segid a ) ( segid b ) WEIGhts * 1. END
    INTEraction ( segid a ) ( segid a ) WEIGhts * 0. bonds 1. END
    INTEraction ( segid b ) ( segid b ) WEIGhts * 0. bonds 1. END
END
```

Three double selections are set up, each with a different weighting scheme. The Hamiltonian is the sum of the three partial Hamiltonians. The result is to turn off the intrasegment interactions other than the bonds.

5 Geometric and Energetic Analysis

5.1 Analysis of Conformational Energy Terms

There are two facilities in XPLOR to analyze the bond, angle, dihedral, improper, and h-bond energy terms. The print statement provides a listing of pertinent information about selected bonds, angles, dihedrals, impropers, and hydrogen bonds. The listing of the print statement can be further reduced by use of the constraints interaction statement (see Section 4.7). The pick statement allows one to pick specific energy terms independent of the actual list of bonds, angles, dihedrals, and impropers in the molecular structure.

5.1.1 Syntax of the Print Statement

PRINt <**print-statement**> is invoked from the main level of X-PLOR. Execution is activated as soon as all required statements are given.

<**print-statement**>:== [**THREshold=**<**real**>] <**print-objects**> prints objects. THREshold is optional (default: THREshold=0).

<**print-objects**>:==

ANGLes lists bond angles that deviate by more than the THREshold value from the equilibrium value in degrees; it also puts the value of the rms deviation into the symbol $RMS and the number of violations into $VIOLATIONS.

BONDs lists bond lengths that deviate by more than the THREshold value from the equilibrium value in Å; it also puts the value of the rms deviation into the symbol $RMS and the number of violations into $VIOLATIONS.

CDIHedrals lists dihedral angle restraints (see Section 7.2) that deviate by more than the THREshold value from the equilibrium value in degrees; it also puts the value of the rms deviation into the symbol $RMS and the number of violations into $VIOLATIONS.

DIHEdrals lists dihedral angles that deviate by more than the THREshold value from the equilibrium value in degrees; it also puts the value of the rms deviation into the symbol $RMS and the number of violations into $VIOLATIONS. Multiple dihedral angle entries ($m > 1$, Eq. 4.6) are characterized by multiple instances of the same four atoms followed by the corresponding parameters for the multiple dihedral.

HBONds lists hydrogen bonds between atoms that have been designated donors and acceptors (DONOr and ACCEptor; see Section 3.1.1) and that are within the cutoff limits specified in the hbonds statement (see Section 3.2.1). This facility can be used even if the current empirical energy function does not use explicit hydrogen-bonded terms. All that is required is to include the ACCEptor and DONOr statements in the residue statement (see Section 3.1.1) and the HBONded statements in the parameter statement.

IMPRopers lists improper angles that deviate by more than the THREshold value from the equilibrium value in degrees; it also puts the value of the rms deviation into the symbol $RMS and the number of violations into $VIOLATIONS.

NOE lists distance restraints (see Chapter 18) that deviate by more than the THREshold value from the equilibrium value in Å; it also puts the value of the rms deviation into the symbol $RMS and the number of violations into $VIOLATIONS. The definition of the deviation depends on which NOE POTEntial is specified: in the case of the SQUAre-well and SOFTsquare potentials, the deviation refers to the difference between the actual distance and the upper or lower bound if the distance is outside the error range; in the case of the BIHArmonic potential, the deviation refers to the difference between the actual distance and the target distance. (A more precise definition is given by Δ in Section 18.3.)

5.1.2 Syntax of the Pick Statement

PICK <**pick-statement**> is invoked from the main level of X-PLOR. Execution is activated as soon as all required statements are given.

<**pick-statement**>:==

ANGLe <**selection**> <**selection**> <**selection**> <**property**> picks bond angle properties between selected atoms. Except for the GEOMetry property, each selection should select exactly one atom; for GEOMetry, the center of mass of each selection is computed, and then the angle between the centers of mass is evaluated. The result is also stored in the symbol $RESULT.

BOND <**selection**> <**selection**> <**property**> picks covalent bond properties between selected atoms. Except for the GEOMetry property,

each selection should select exactly one atom; for GEOMetry, the center of mass of each selection is computed, and then the bond between the centers of mass is evaluated. The result is also stored in the symbol $RESULT.

DIHEdral <**selection**> <**selection**> <**selection**> <**selection**> <**property**> picks dihedral angle properties between selected atoms. Except for the GEOMetry property, each selection should select exactly one atom; for GEOMetry, the center of mass of each selection is computed, and then the dihedral angle between the centers of mass is evaluated. The result is also stored in the symbol $RESULT.

IMPRoper <**selection**> <**selection**> <**selection**> <**selection**> <**property**> picks improper properties between selected atoms. Except for the GEOMetry property, each selection should select exactly one atom; for GEOMetry, the center of mass of each selection is computed, and then the improper angle between the centers of mass is evaluated. The result is also stored in the symbol $RESULT.

RING { <**selection**> } <**property**> picks the generalized pucker parameter of the selected ring as described by Cremer and Pople (1975). The ring is selected by specifying all atoms by multiple selection (Section 2.15). Each selection should refer to exactly one atom. At least four selections are required. There are $q = n/2 - 1$ amplitudes and $p = (n-1)/2 - 1$ phases where n is the number of atoms of the ring. The amplitudes are stored in the symbols $AMPLITUDE1, $AMPLITUDE2, ...and the phases are stored in the symbols $PHASE1, $PHASE2, ... The number of amplitudes q is stored in the symbol $N_AMPLITUDE and the number of phases p is stored in the symbol $N_PHASE.

<**property**>:==

ENERgy is the potential energy value of the corresponding partial energy term (does not apply to RING).

GEOMetry supplies information about the actual value of the picked object (does not apply to RING).

PUCKer applies only to RING.

5.1.3 Requirements

The molecular structure, coordinates, and corresponding parameters have to be well defined for the selected energy terms. For the GEOMetry property, no parameters are required.

5.1.4 Example: Print Bond Length and Bond Angle Deviations

The following example shows how to print all covalent bond lengths and angles that deviate respectively by more than 0.1 Å and 10.0° from the equilibrium values. It also provides information about the rms deviation of the selected bonds and angles.

```
print threshold=0.1 bonds
print threshold=10.0 angles
```

5.1.5 Example: Print All Bonds of a Selected Set of Atoms

This example shows how to print all covalent bonds of residue 40 that deviate by more than 0.1 Å. It also provides information about the rms deviation of the selected bonds.

```
constraints interaction
  ( residue 40 ) ( residue 40 )
end
print threshold=0.1 bonds
```

5.1.6 Example: Pick Bond Geometry

This example picks the bond length between the "c" and the "o" atom of residue 1:

```
pick bond
   ( resid 1 and name c ) ( resid 1 and name o ) geometry
end
```

5.1.7 Example: Pick Distance between Two Atoms

This example picks the distance between two atoms. No parameters are required, and the atoms do not have to be bonded to each other.

```
pick bond
   ( resid 5 and name nz ) ( resid 1 and name o ) geometry
end
```

5.1.8 Example: Pick Bond Angle among Three Atoms

This example picks the bond angle value among three arbitrary atoms. Note that the atoms do not have to be bonded to each other, and no parameters are required for this property. Of course, if one tried to pick a different property here, the program would come back with an error message.

```
pick angle
   ( residue 1 and name c )
   ( residue 32 and name n )
   ( residue 5 and name ca )
   geometry
```

5.2 Analysis of the Nonbonded Energy Terms

The distance statement allows one to analyze the nonbonded interactions, such as inter- or intramolecular close contacts. The nonbonded distances are analyzed as provided by the nonbonded list routines. Selected parts of the nonbonded list may be printed by specifying an upper and lower cutoff and atom selections. All nonbonded distances between the first and second set of atoms that satisfy the cutoff criteria are printed. (See Section 3.2.1 for nonbonded list parameters and options.) It should be noted that the distance statement is most useful for checking contacts between molecules. It can also be used to produce a distance matrix. However, the distance statement is not very flexible when certain averages of distances are required. (See Section 6.4 for a much more flexible approach to computing and plotting a distance matrix.)

5.2.1 Syntax

DISTance { <**distance-statement**> } is invoked from the main level of X-PLOR. The END statement activates execution.

<**distance-statement**>:==

CUTOFf=<real> is an upper distance cutoff: distances less than CUTOff and less than the list cutoff (CUTNB) are analyzed.

CUTON=<real> is a lower distance cutoff.

DISPosition=<MATRix|PRINt|RMSD> specifies how distances will be stored or printed. RMSD stores the minimum nonbonded distance for each atom in the first atom selection to all atoms in the second atom selection in the RMSD atom property. PRINt writes all selected nonbonded distances to standard output. MATRix stores all selected, nonbonded distances in a matrix and then writes the matrix to the specified output file.

FROM=<selection> specifies the first atom selection (default: (ALL)).

OUTPut=<filename> specifies a file for the distance matrix (only for DISPosition=MATRix) (default: OUTPUT).

TO=<selection> specifies the second atom selection.

5.2.2 Requirements

The molecular structure, coordinates, and parameters (especially the nonbonded parameters) have to be defined.

5.2.3 Example: Distances between Selected Residues

In the following example, all nonbonded distances between residue 10 and residue 70 that are within the nonbonded cutoff (CUTNB) as well as within the cutoff (CUTOFF) of the distance statement are printed.

```
parameter
  nbonds cutnb=20.
end
distance
  from=( residue 10 )
  to=( residue 70 )
  cuton=0.
  cutoff=20.
end
```

5.3 Deviations from Ideality and Crystal Packing

The following example file shows how to analyze the rms deviations from ideality for bonds and bond, dihedral, and improper angles as well as nonbonded contacts. In this particular example crystallographic symmetry is present, and the crystal packing contacts are analyzed as well.

```
remarks   file  geomanal/geomanal.inp
remarks   Analyze geometry and nonbonded contacts

                                                        {*Read structure file.*}
{===>} structure @../generate/generate.psf end

                                                           {*Read coordinates.*}
{===>} coordinates @../xtalrefine/slowcool.pdb

{===>} vector ident (store9) ( not hydrogen )
                                              {*This selects all atoms to be*}
                                              {*analyzed.  Change it        *}
                                              {*to analyze portions,        *}
                                              {*e.g., ( segid "A" ).        *}


parameter
   {===>} @TOPPAR:parhcsdx.pro          {*Read empirical energy parameters.*}
end


xrefine                          {*Unit cell parameters and crystal symmetry.*}

{===>}   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0

{===>}   symmetry=(x,y,z)
         symmetry=(-x+1/2,-y,z+1/2)
         symmetry=(-x,y+1/2,-z+1/2)
         symmetry=(x+1/2,-y+1/2,-z)
end


           {*Deviations of the geometry*}
           {*==========================*}

constraints interaction                       {*This command selects the   *}
    ( store9 ) ( store9 )                     {*atoms to be                *}
```

```
end                                          {*considered in the following*}
                                             {*analysis.                  *}

print threshold=0.06 bonds                   {*Lists bonds deviating more   *}
                                             {*than 0.06 A from ideality.   *}

print threshold=10.0 angles                 {*Lists angles deviating more   *}
                                            {*than 10 degrees from ideality.*}

print threshold=90.0 dihedrals              {*Lists dihedrals deviating more*}
                                            {*than 90 degrees from ideality.*}

print threshold=20.0 impropers              {*Lists impropers deviating more*}
                                            {*than 20 degrees from ideality.*}

          {*bad? contacts and nonbonded energy*}
          {*==================================*}
flags exclude * include vdw elec pvdw pele end
energy end

          {*crystal packing contacts*}
          {*========================*}
flags exclude * include pvdw end


distance from=( not hydrogen ) to=( not hydrogen ) cutoff=4.0 end

stop
```

5.4 Conformation vs. Residue Number

The following example file shows how to produce a print or plot file that indicates the deviations of bonds and bond, dihedral, and improper angles from ideality as a function of residue number. This information is useful for crystallographic refinement (Chapter 13) as well as NMR structure refinement (Chapter 18). It can be used to localize problems in the refined coordinates.

```
remarks file geomanal/geomplot.inp -- Give conformational energy per residue
remarks Provide a printer output file as well as a list file
remarks for plotting

                                                    {*Read structure file.*}
  {===>} structure @../generate/generate.psf end

                                                       {*Read coordinates.*}
  {===>} coordinates @../xtalrefine/slowcool.pdb

  {===>} vector ident (store9) ( tag )    {*This selects all residues to   *}
                                          {*be analyzed.   Change it       *}
                                          {*to analyze portions,           *}
                                          {*e.g.,  ( tag and segid "A" ).  *}
                                          {*"tag" assigns one unique atom  *}
                                          {*per residue  (see <selection>).*}


  parameter
   {===>} @TOPPAR:parhcsdx.pro          {*Read empirical energy parameters.*}
  end
```

```
                                          {*Use only conformational energy.*}
flags exclude * include bond angl dihe impr end


set echo=off end                          {*Turn off echo to reduce output.*}
set message=off end                           {*Turn off warning messages.*}

                                            {*Calculate energy per residue.*}

for $atom_id in id ( store9 ) loop main
    constraints interaction ( byresidue ( id $atom_id ) ) ( all ) end
    energy end
    vector do ( store1=$ener ) ( id $atom_id )
end loop main

                                      {*Calculate E / rms(E) for each residue.*}

vector show rms (store1) ( store9 )
vector do (store1=store1/$result) ( store9 )


{===>} set display=geomplot.print end   {*Write a histogram of energy to *}
                                       {*the file named "geomplot.plot".*}



for $atom_id in id ( store9 ) loop outp
    vector show element (store1) ( id $atom_id )
    if ($result < 1. ) then
       eval ($graph = " " )
    elseif ($result < 2. ) then
         eval ($graph = "X" )
    elseif ($result < 3. ) then
         eval ($graph = "X-X" )
    elseif ($result < 4. ) then
         eval ($graph = "X-X-X" )
    elseif ($result < 5. ) then
         eval ($graph = "X-X-X-X" )
    elseif ($result < 6. ) then
         eval ($graph = "X-X-X-X-X" )
    elseif ($result < 7. ) then
         eval ($graph = "X-X-X-X-X-X" )
    elseif ($result < 8. ) then
         eval ($graph = "X-X-X-X-X-X-X" )
    elseif ($result < 9. ) then
         eval ($graph = "X-X-X-X-X-X-X-X" )
    elseif ($result  < 10. ) then
         eval ($graph = "X-X-X-X-X-X-X-X-X" )
    else
         eval ($graph = "X-X-X-X-X-X-X-X-X-X-->" )
    end if
    vector show elem ( resid ) ( id $atom_id )
    evaluate ( $resid=$result )
    vector show elem ( segid ) ( id $atom_id )
    evaluate ( $segid=$result )
    display  $segid  $resid   $graph
end loop outp

{===>} set display=geomplot.list end           {*Write a list file to the*}
                                              {*specified file.         *}

evaluate ($number=0)
for $atom_id in id ( store9 ) loop out2
    vector show element (store1) ( id $atom_id )
    evaluate ( $arg=$result )
    evaluate ($number=$number+1)
    display   $number  $arg
end loop out2

set echo=on end
```

```
96   set message=on end
97
98   stop
99
```

The following Mathematica file was used to produce the plot shown in Fig. 5.1. The information is plotted as a function of sequential residue number.

```
1         (* file geomanal/geomplot.math  -- Make a plot rmsd vs residue     *)
2
3  Off[General::spell1];
4  $DefaultFont={"Times-Roman",13};
5
6                                      (* Change the name of the input file. *)
7   (*===>*)    data=ReadList["geomplot.list",{Number,Number}];
8
9       column1 = Transpose[data][[1]];
10      column2 = Transpose[data][[2]];
11      max=Max[column2];
12      number=Dimensions[column1][[1]];
13      ListPlot[column2,
14       AxesLabel->{"Seq. Res. No.","Geometry"},
15       Ticks->{Range[0, number , 5],Range[0.0, max ,1.]},
16       PlotLabel->"GeomPlot", AxesOrigin->{0,0},PlotJoined->True,
17       PlotRange->{{0, number +2 }, {0., max +2 }} ];
18
```

The resulting plot looks like this:

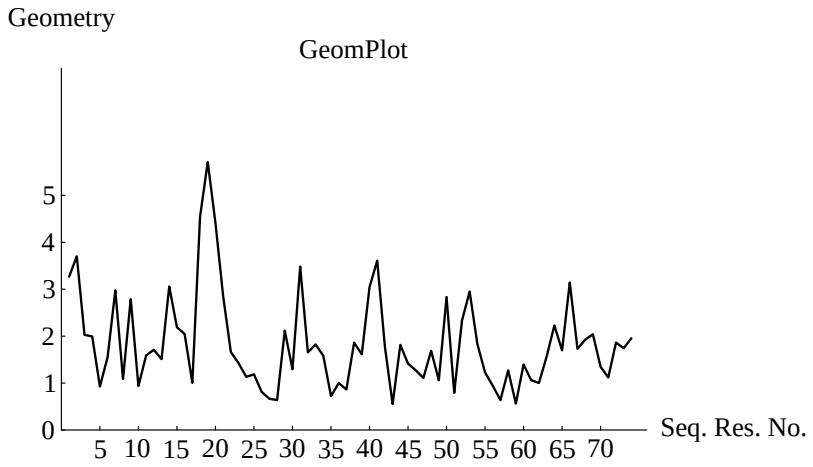

Figure 5.1: Geometry *vs.* residue number.

The correspondence between sequential residue number and actual residue identifier can be established by issuing the following statements:

```
1 vector ident (store1) ( tag )
2 vector show element ( store1 ) ( attr store1 > 0 )
```

A particular residue identifier can be queried by saying

```
vector ident (store1) ( tag )
vector show element ( store1 ) ( attr store1 = 41 )
```

which will show the residue identifier that corresponds to the 41st sequential residue.

5.5 Ramachandran Plot

The following example shows how to make a Ramachandran plot:

```
remarks file geomanal/ramachandran.inp -- Make a Ramachandran (phi-psi) plot
remarks      (this file is specific for proteins)

                                                    {*Read structure file.*}
{===>} structure @../generate/generate.psf end

                                                       {*Read coordinates.*}
{===>} coordinates @../xtalrefine/slowcool.pdb

parameter
   {===>} @TOPPAR:parhcsdx.pro          {*Read empirical energy parameters.*}
end

{===>} vector ident (store9) ( name ca )     {*This selects all residues to*}
                                             {*be analyzed.   Change it    *}
                                             {*to analyze portions, e.g.,  *}
                                             {*( name ca and segid "A" ).  *}
                                           {*"tag" assigns one unique atom *}
                                           {*per residue (see <selection>).*}

{===>} set display=ramachandran.list end    {*Write the phi psi info to the*}
                                            {*specified file.              *}

set echo=off end                          {*Turn off echo to reduce output.*}
set message=off end                       {*Turn off warning messages.     *}


evaluate ($pre_atom_id=0)                {*Initialize atom ids for residue *}
evaluate ($pre_pre_atom_id=0)            {*n-1 and n-2.                    *}

for $atom_id in id ( name ca ) loop main          {*Loop over all ca atoms.*}
    if ( $pre_pre_atom_id > 0 ) then


        pick  dihedral                                          {*Pick phi.*}
            ( byresidue ( id $pre_pre_atom_id ) and name C  )
            ( byresidue ( id $pre_atom_id ) and name N  )
            ( byresidue ( id $pre_atom_id ) and name CA  )
            ( byresidue ( id $pre_atom_id ) and name C  )
             geometry
        evaluate ($phi=$result)

        pick  dihedral                                          {*Pick psi.*}
            ( byresidue ( id $pre_atom_id ) and name N  )
            ( byresidue ( id $pre_atom_id ) and name CA  )
            ( byresidue ( id $pre_atom_id ) and name C  )
            ( byresidue ( id $atom_id ) and name N  )
             geometry
        evaluate ($psi=$result)

        vector show element ( resname ) ( byresidue ( id $pre_atom_id )
                                        and name ca )
        evaluate ( $resname=$result )

                                          {*Depending on the residue name,*}
```

```
                                          {* write a "1" or a "0" .       *}
        if ($resname = "GLY" ) then
           display    0  $phi $psi
        else
           display    1  $phi $psi
        end if

    end if
    evaluate ($pre_pre_atom_id=$pre_atom_id)          {*Reassign atom ids.*}
    evaluate ($pre_atom_id=$atom_id)
end loop main


stop
```

X-PLOR produces the file ramachandran.list, which can be interpreted by Mathematica using the following file:

```
           (* file geomanal/ramachandran.math  -- Make a Ramachandran plot *)

Off[General::spell1];
$DefaultFont={"Times-Roman",13};

                                    (* Change the name of the input file. *)
 (*===>*)
  data=ReadList["ramachandran.list",{Number,Number,Number}];

  column1 = Transpose[data][[1]];
  column2 = Transpose[data][[2]];
  column3 = Transpose[data][[3]];

  number=Dimensions[column1][[1]];

            (* If it is a GLY residue, X-PLOR wrote a 0 and we make a "o"; *)
            (* otherwise we make an "x".                                    *)
  Data=Table[ Which[ column1[[i]]==1,
                     Text[FontForm[x, {"Plain", 6}],
                       { column2[[i]], column3[[i]] } ] ,
                   column1[[i]]==0,
                     Text[FontForm[o, {"Plain", 6}],
                       { column2[[i]], column3[[i]] } ] ],
          { i,  number }];

                   (* Make the "allowed" regions of the Ramachandran plot. *)
 Data=Join[Data,{
  Line[{{-156.5,91.3},{-70.4,91.3},{-54.7,112.8},{-54.7,173.2},
       {-136.95,173.2},{-136.9,155.8},{-156.5,135.6},{-156.5,91.3}}],
  Line[{{-180.,42.9},{-140.8,16.1},{-86.0,16.1},{-74.3,45.6},
       {-74.3,72.5},{-44.3,102.0},{-44.3,161.1},{-46.9,179.9}}],
  Line[{{-180.,-34.9},{-164.3,-42.9},{-133.0,-42.9},{-109.5,-32.2},
       {-106.9,-21.4},{-44.3,-21.4},{-44.3,-71.1},{-180.,-71.1}}],
  Line[{{-156.5,-60.4},{-54.7,-60.4},{-54.7,-40.2},{-100.4,-40.2},
       {-123.9,-51.0},{-156.5,-51.0},{-156.5,-60.4}}],
  Line[{{-180.,-163.8},{-75.6,-163.8},{-46.9,-180.}}],
  Line[{{62.6,14.7},{62.6,96.7},{45.6,79.2},{45.6,26.8},{62.6,14.7}}]
       }] ;

  Show[Graphics[ Data,
    Ticks->{Range[-180.,180.,60.],Range[-180.,180.,60.]},
    AxesOrigin->{-180.,-180.},
    PlotRange->{{-180., +180.}, {-180., +180.}},
    AspectRatio->1,
    Frame->True,
    Axes->True,
    AxesLabel->{"phi","psi"},PlotLabel->"Ramachandran Plot"
    ]];
```

The resulting plot is shown in Fig. 5.2.

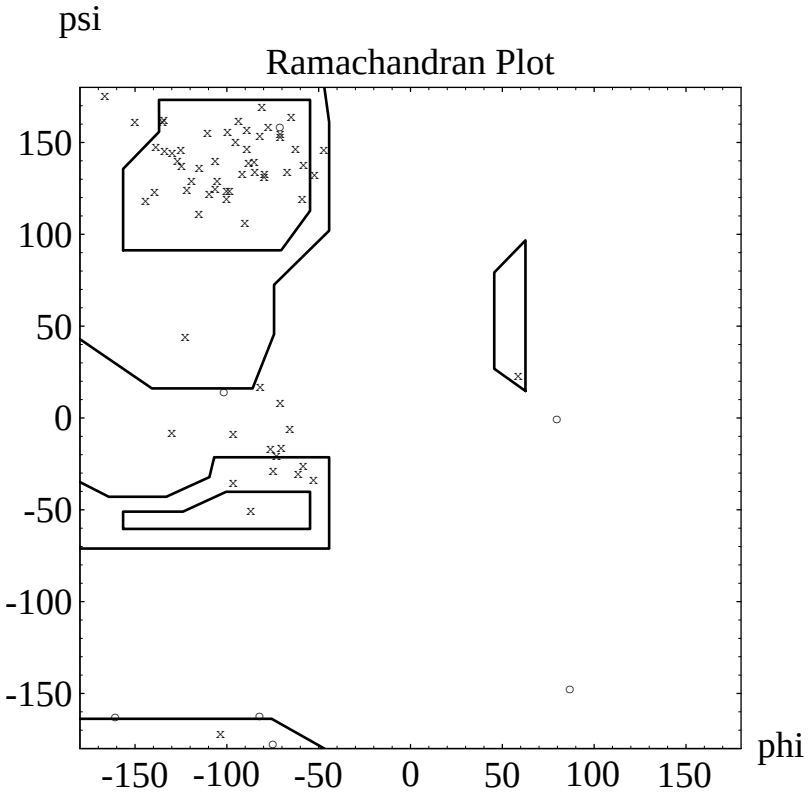

Figure 5.2: Ramachandran plot.

5.6 Accessible Surface Area

The algorithm by Lee and Richards (1971) is used to compute the accessible surface area (MODE=ACCEss) or the contact area (MODE=CONTact). The routine uses half the σ values (RADIus=SIGMa) or half the van der Waals radii (RADIus=VDW) (Eq. 4.8 and 4.10 for computing the solvent excluded volume.

The contact area is obtained from the accessible surface area by the following formula:

$$(\text{atom\_radius}/(\text{atom\_radius} + \text{water\_radius})^2 \tag{5.1}$$

Upon completion, the accessible surface or contact area for each selected atom is stored in the RMSD atom property (in Å^2 units).

5.6.1 Syntax

SURFace { <**surface-statement**> } **END** is invoked from the main level of X-PLOR. The END statement activates execution.

<**surface-statement**>:==

ACCUracy=<real> is accuracy of the numerical integration (default: 0.05).

MODE=ACCess | CONTact is access or contact mode (default: access).

RADIus=SIGMa | VDW specifies whether σ values or van der Waals radii are used (default: SIGMa).

RH2O=<real> is probe radius (default: 1.6 Å).

SELEction=<selection> performs the calculation for the selected atoms.

5.6.2 Requirements

The molecular structure, coordinates, and parameters have to be defined.

5.6.3 Example

Here the accessible surface area is computed and printed. The results can also be written to a Mathematica file for plotting (see the rmsd plot in Section 6.3).

```
1 surface
2    rh2o=1.6
3    mode=access
4 end
5 vector show elem ( rmsd ) ( not hydrogen )
```

6 Cartesian Coordinates

6.1 Coordinate Statement

The coordinate statement is used to read and manipulate coordinates, such as least-squares fitting to a comparison coordinate set (Kabsch 1976), rotation, translation, and conversion between fractional and orthogonal coordinates.

6.1.1 Syntax

The coordinate statement options are carried out when the END statement is issued. If one wants to carry out several coordinate manipulations, each manipulation has to be initiated separately from the main level of X-PLOR.

COORdinates <coordinate-statement> END is invoked from the main level of X-PLOR. The END statement activates execution of the particular operation.

<coordinate-statement>:==

COPY [SELEction=<selection>] copies main coordinate set into comparison set; XCOMP:=X,YCOMP:=Y,ZCOMP:=Z; B,Q are unaffected (default for selection: (ALL)).

FIT { [SELEction=<selection>] [MASS=<logical>] [LSQ=<logical>] } rotates (if LSQ is TRUE) and translates all main coordinates to obtain a best fit between the selected main and comparison atoms. Translation superimposes the geometric centers (or the centers of mass if MASS is TRUE) of the two coordinate sets. Rotation occurs with the Kabsch (1976) least-squares fitting algorithm. Mass-weighting is applied if MASS is set to TRUE. Upon successful completion of this operation, the Eulerian angles describing the rotation matrix R are stored in the symbols $THETA1, $THETA2, $THETA3, the rotation axes in the symbols $AXIS_X, $AXIS_Y, $AXIS_Z, $AXIS_KAPPA, the spherical polars in the symbols $PSI, $PHI, $KAPPA, the rotation matrix in the symbols $ROT_1_1, ..., $ROT_3_3, and the translation vector T is stored in the symbols $X, $Y, $Z. The fitted coordinate set r' is related to

original set r by $r' = R * r + T$ (default for selection: (ALL); for MASS: FALSE; for LSQ: TRUE).

FRACtionalize { **[SELEction=<selection>]**
[A=<real>] [B=<real>] [C=<real>]
[ALPHa=<real>] [BETA=<real>] [GAMMa=<real>] } fractionalizes selected coordinates. The X-PLOR convention keeps the direction of the x-axis (x same direction as a; y is in (a,b) plane) and is the same that is used internally by all X-PLOR routines (default for selected atoms: (ALL); for a, b, c: 1.0; for α, β, γ: 90°).

INITialize { **[SELEction=<selection>]** } initializes main coordinate set; i.e., X:=9999.0, Y:=9999.0, Z:=9999.0; B,Q are unaffected (default for selection: (ALL)).

ORIEnt { **[SELEction=<selection>] [MASS=<logical>]**
[LSQ=<logical>] } rotates (if LSQ is TRUE) and translates all coordinates such that the principal axis system of the selected atoms corresponds to the x,y,z axis. The translation superimposes the geometric center (or the center of mass if MASS is TRUE) and the coordinate origin. The rotation occurs with the Kabsch (1976) least-squares fitting algorithm. Mass-weighting is applied if MASS is set to TRUE. Upon successful completion of this operation, the Eulerian angles describing the rotation matrix R are stored in the symbols $THETA1, $THETA2, $THETA3, the rotation axes in the symbols $AXIS_X, $AXIS_Y, $AXIS_Z, $AXIS_KAPPA, the spherical polars in the symbols $PSI, $PHI, $KAPPA, the rotation matrix in the symbols $ROT_1_1, ..., $ROT_3_3, and the translation vector T is stored in the symbols $X, $Y, $Z. The oriented coordinate set r' is related to original set r by $r' = R * r + T$ (default for selection: (ALL); for MASS: FALSE; for LSQ: TRUE).

ORTHogonalize { **[SELEction=<selection>]**
[A=<real>] [B=<real>] [C=<real>]
[ALPHa=<real>] [BETA=<real>] [GAMMa=<real>] } orthogonalizes selected coordinates. The X-PLOR convention keeps the direction of the x-axis (x same direction as a; y is in (a,b) plane) and is the same that is used internally by all X-PLOR routines (default for selected atoms: (ALL); for a, b, c: 1.0; for α, β, γ: 90°).

RGYRation { **[SELEction=<selection>] [MASS=<logical>]**
[FACT=<real>] } computes radius of gyration

$$R_{gyr} = \sqrt{< (r_i - < r_i >)^2 >} \tag{6.1}$$

where the angle brackets denote averaging over selected atoms. The averaging is mass-weighted if MASS is TRUE. The factor FACT is subtracted from the masses before applying the mass-weighting. The symbols $RG (radius of gyration), $XCM, $YCM, $ZCM (center of mass) are declared (default for selected atoms: (ALL); for MASS: FALSE; for FACT: 0.0).

RMS { **[SELEction** <**selection**>] **[MASS=**<**logical**>] } computes the (mass-weighted if MASS is TRUE) rms difference for selected atoms between the main and comparison set. The rms value is stored in the symbol $RESULT. The individual atomic rms differences are stored in the RMSD array (default for selection: (ALL); for MASS: FALSE).

ROTAte { **[SELEction=**<**selection**>] **[CENTer=**<**3d-vector**>] <**matrix**> } rotates selected atoms around the specified rotation center (default: (0 0 0)). The rotation matrix is specified through the matrix statement (see Section 2.4). Upon successful completion of this operation, the Eulerian angles describing the rotation matrix R are stored in the symbols $THETA1, $THETA2, $THETA3, the rotation axes in the symbols $AXIS_X, $AXIS_Y, $AXIS_Z, $AXIS_KAPPA, the spherical polars in the symbols $PSI, $PHI, $KAPPA, the rotation matrix in the symbols $ROT_1_1, ..., $ROT_3_3, and the translation vector T is stored in the symbols $X, $Y, $Z. (default for selection: (ALL)).

SHAKe { **[MASS=**<**logical**>] **[REFErence=MAIN|COMP]** } iteratively modifies main coordinate set until SHAKE constraints (see Section 8.2) are satisfied. The reference coordinate set specifies the direction of the SHAKE shift vectors (default for MASS: FALSE; for REFErence: MAIN).

SWAP { **[SELEction=**<**selection**>] } exchanges main and comparison coordinate set, i.e., X,Y,Z $\leftrightarrow$ XCOMP,YCOMP,ZCOMP; B,Q are unaffected (default for selection: (ALL)).

SYMMetry <**symmetry-operator**> **[SELEction=**<**selection**>] applies the specified crystallographic symmetry operator to the selected coordinates. The notation for the symmetry operator is the same as in the *International Tables for Crystallography* (Hahn ed. 1987), e.g., $(-x, y + 1/2, -z)$. The unit cell geometry needs to be specified (Section 12.3) before invoking this statement. The coordinates are converted into fractional coordinates before application of the symmetry operator and converted back into orthogonal coordinates afterward.

TRANslate { **[SELEction=**<**selection**>] **VECTor=**<**3d-vector**> **[DISTance=**<**real**>] } translates selected atoms by specified translation vector. If DISTance is specified, the translation occurs along the specified vector for the specified distance (default for selection: (ALL)).

COOR <**coordinate-read-statement**> **END** reads coordinates. It is invoked from the main level of X-PLOR.

<**coordinate-read-statement**>**:==** **[DISPosition= COMParison | DERIvative | MAIN | REFErence] [SELEction=**<**selection**>] { <**pdb-record**> } reads Brookhaven Data Bank formatted records consisting of x,y,z coordinates, occupancies, and B-factors, tries to match the atom name, residue name, residue number, and segment name, and

deposits the information in the main (X,Y,Z,B,Q), comparison (XCOMP, YCOMP, ZCOMP, BCOMP, QCOMP), reference (REFX, REFY, REFZ, HARM, HARM), or derivative (DX, DY, DZ, FBETA, FBETA) coordinate arrays. (For the definition of these arrays, see Section 2.16.) The information is deposited only if the atom has been selected. Note that the syntax is strict; i.e., the SELEction and the DISPosition have to be specified before one can read a <pdb-record>. Also, the PDB convention suggests an END statement at the end of the file that will terminate the coordinate statement (default for selected atoms: (ALL); for DISPosition: MAIN).

<pdb-record>:== According to the Brookhaven Protein Data Bank, the entry for atoms is defined as follows:

```
1 ATOM    837 HG23 THR  1055      -8.573   5.657  -3.818  1.00  0.00
2 ATOM   1223  O   GLY   153A    -11.704  -9.200    .489  1.00  0.80
3       uuuuu vvvv uuuuCuuuuI   vvvvvvvvvuuuuuuuuuvvvvvvvvvuuuuuuuvvvvvv iii
4         atom     residue          x       y        z     q      b    entry#
5     number name name number
6                          ^ insertion character
7                     ^ chain identifier
8             ^ additional character for some atom names (mostly h's)
```

X-PLOR does not use the chain identifier information. Instead, it uses the characters in columns 73–76 for the segment name (see Section 3.7). The segment name has to match the definition in the segment statement. The insertion character is treated as part of the residue number (note: the residue number is a string consisting of a maximum of four characters). X-PLOR ignores any reference to the atom numbers and instead generates its own numbering scheme. The REMARK record of PDB files is treated as a title record (cf. REMARKS, Section 2.8). No other type of PDB specification, such as HETAT, SCALE, or SEQU, is interpreted at present. These additional records have to be removed before one reads PDB coordinates with X-PLOR. Initially, the user should divide the original PDB file into files containing individual protein chains, individual substrates, all waters combined, and individual cofactors. This will simplify the molecular structure generation with X-PLOR (see Section 3.7). Normally, X-PLOR expects orthogonal coordinates. The ORTHogonalize option can be used to convert fractional coordinates into orthogonal Å coordinates (see Section 12.2).

The PDB convention requires an END statement at the end of the coordinate file. X-PLOR uses the same convention. The inclusion of the END statement implies that the coordinate statement must not be terminated with an END statement from the main level of X-PLOR. However, if the END statement is missing in the coordinate file, parsing errors will result.

6.1.2 Requirements

The molecular structure has to be defined before any coordinates can be read and/or manipulated.

6.1.3 Examples

The first example reads the coordinates from two files into the main and comparison sets:

```
coordinates @set1.pdb
coordinates disposition=comparison  @set2.pdb
```

In the second example, the first set is least-squares fitted to the second (comparison) set using C^{α} atoms only. Note that all atoms are translated and rotated. Then the rms difference between the two sets is computed for backbone atoms and stored in the symbol $1. Finally, the individual rms differences are printed for all backbone atoms that show rms differences greater than 1 Å.

```
coordinates fit selection=( name ca ) end
coordinates rms selection=( name ca or name n or name c ) end
evaluate ($1=$result)
vector show ( rmsd )
     ( attribute rmsd > 1.0 and ( name ca or name n or name c ))
```

In the third example, the main coordinates are fractionalized, a translation and a rotation are applied, and the coordinates are orthogonalized again. The crystallographer will recognize the rotation and translation as the space-group operator $(-x, y + 1/2, -z)$.

```
coordinates fractionalize
   a=30. b=40. c=20. alpha=90. beta=100. gamma=90.
end
coordinates rotate
   matrix=( -1 0 0  )
          (  0 1 0  )
          (  0 0 -1 )
end
coordinates translate
   vector=( 0 0.5 0 )
end
coordinates orthogonalize
   a=30. b=40. c=20. alpha=90. beta=100. gamma=90.
end
```

The final example shows how to rotate the coordinates using Eulerian angles:

```
coordinates rotate
    euler=( 10., 30., 2. )
end
```

6.2 Write Coordinate Statement

The write coordinate statement writes the current coordinates to the specified file. This statement writes an explicit END statement at the end of the file.

6.2.1 Syntax

WRITe COORdinates { <**write-coordinate-statement**> } **END** is invoked from the main level of X-PLOR. The END statement activates writing of coordinates.

<**write-coordinate-statement**>:==

FROM= MAIN | COMParison | REFErence | DERIvative determines which combination of atomic properties is written to the file. MAIN corresponds to X, Y, Z, B, Q; COMParison corresponds to XCOMP, YCOMP, ZCOMP, BCOMP, QCOMP; REFErence corresponds to REFX, REFY, REFZ, HARM, HARM; and DERIvative corresponds to DX, DY, DZ, FBETA, FBETA (default: MAIN).

OUTPut=<**filename**> specifies the output filename (default: OUTPUT).

SELEction=<**selection**> writes coordinates of selected atoms to the file (default: (ALL)).

6.2.2 Requirements

The molecular structure and the appropriate coordinate arrays have to be defined.

6.2.3 Example

The following example shows how to write the main coordinate arrays to the specified file using C^{α} atoms only:

```
write coordinates
   from=main
   output=file.pdb
   selection=( name ca )
end
```

6.3 Rms Differences between Coordinates

The following example shows how to compute all rms differences (rmsd) between two coordinate sets (one in the main set, one in the comparison set). The rmsds are rms-averaged over all atoms of each residue. The user is also referred to Sections 20.8 and 20.9 for computation of averages and rmsds of a family of coordinate sets.

```
remarks file geomanal/rmsdplot.inp -- Rms difference per residue
remarks Specific for proteins: backbone and sidechain atoms.

                                                {*Read structure file.*}
  {===>} structure @../generate/generate.psf end

```

```
                                                    {*Read coordinates.*}
{===>} coordinates @../xtalrefine/slowcool.pdb

                                                {*Read comparison coords.*}
{===>} coordinates disposition=comp @../generate/generate.pdb

{===>} vector ident (store9) ( tag )     {*This selects all residues to  *}
                                         {*be analyzed.  Change it       *}
                                         {*to analyze portions,          *}
                                         {*e.g.,  ( tag and segid "A" ). *}
                                         {*"tag" assigns one unique atom *}
                                         {*per residue (see <selection>).*}

coor fit selection=( store9 ) end {*Fit coordinates using selected atoms.*}

                     {* swap equivalent groups to minimize rms difference *}
@rotares.inp

coor rms end                      {*Compute rmsds and store in RMSD array.*}

{===>} set display=rmsdplot.list end   {*Write the rmsds to the specified*}
                                       {*file.                           *}

set echo=off end                        {*Turn off echo to reduce output.*}
set message=off end                          {*Turn off warning messages.*}

evaluate ( $number=0 )
for $atom_id in id ( store9 ) loop out2
    vector show norm ( rmsd ) ( byresidue ( id $atom_id )
                              and ( name ca or name n or name c ) )
    evaluate ( $back=$result )
    vector show norm ( rmsd ) ( byresidue ( id $atom_id )
                              and not ( name ca or name n or name c )
                              and not hydrogen )
    evaluate ( $side=$result )
    evaluate ($number=$number+1)
    display  $number  $back   $side
end loop out2


set echo=on end
set message=on end
stop
```

X-PLOR produces the file "rmsdplot.list", which can be interpreted by Mathematica:

```
      (* file geomanal/rmsdplot.math  -- Make a plot rmsd vs residue.     *)

Off[General::spell1];
$DefaultFont={"Times-Roman",13};

                                  (* Change the name of the input file. *)
 (*===>*)    data=ReadList["rmsdplot.list",{Number,Number,Number}];

    column0 = Transpose[data][[1]];
    column1 = Transpose[data][[2]];
    column2 = Transpose[data][[3]];
    max=Max[column1,column2];
    number=Dimensions[column1][[1]];
    ListPlot[column1,
      Axes->True,
      AxesLabel->{"Seq. Res. No.","RMSD"},
      Ticks->{Range[0, number , 5],Range[0.0, max ,0.2]},
      PlotLabel->"backbone", AxesOrigin->{0,0},PlotJoined->True,
      PlotRange->{{0, number +2 }, {0., max +0.1 }}];
    ListPlot[column2,
      Axes->True,
```

```
22        AxesLabel->{"Seq. Res. No.","RMSD"},
23        Ticks->{Range[0, number , 5],Range[0.0, max ,0.2]},
24        PlotLabel->"side chain", AxesOrigin->{0,0},PlotJoined->True,
25        PlotRange->{{0, number +2 }, {0., max +0.1 }}];
26
```

The resulting plots of the rmsds as a function of sequential residue number are shown in Figs. 6.1 and 6.2.

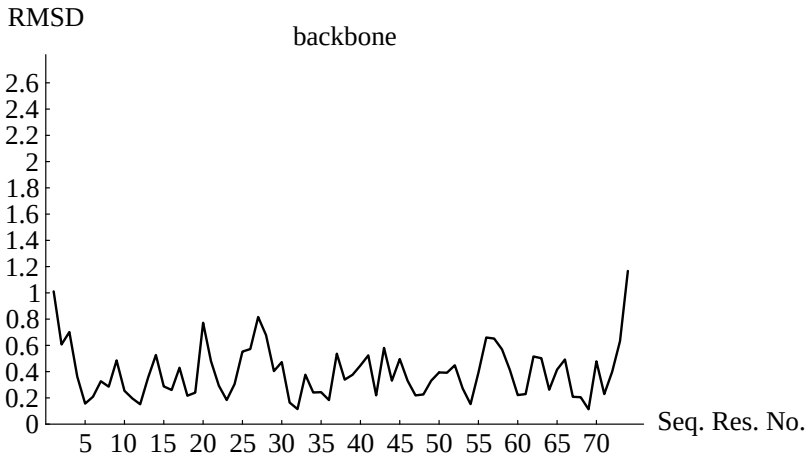

Figure 6.1: Rmsd vs. residue number.

The correspondence between sequential residue number and actual residue identifier can be established by using vector statements (see Section 5.4).

The "rmsdplot.inp" file performs a systematic check for equivalent atoms in the case Tyr, Phe, Glu, Asp, or Arg sidechains. For example, in the case of Tyr, the two orientations of the tyrosine ring are indistinguishable. However, the two orientations show a large difference in terms of r.m.s. devations. The program checks both orientations and picks the one that show the smallest r.m.s. difference. The follwing is the listing of the file "rotates.inp" which contains the statements that accomplish this fit of equivalent atoms.

```
1  REMARK SWAP.INP
2  REMARK swaps specified atoms and checks if rms is decreased
3  REMARK Author: M. Nilges
4
5  set message off echo off end
6
7  evaluate ($oldrms=0)
8  evaluate ($newrms=0)
9
10 for $id in id ( (resn TYR or resn PHE or resn GLU) and name ca ) loop rotr
11
12   vector show element (resid) ( id $id )
13   evaluate ($resid = $result)
14
15   coor rms end
16   evaluate ($oldrms=$result)
17
18   coor rotate sele= (resid $resid and
```

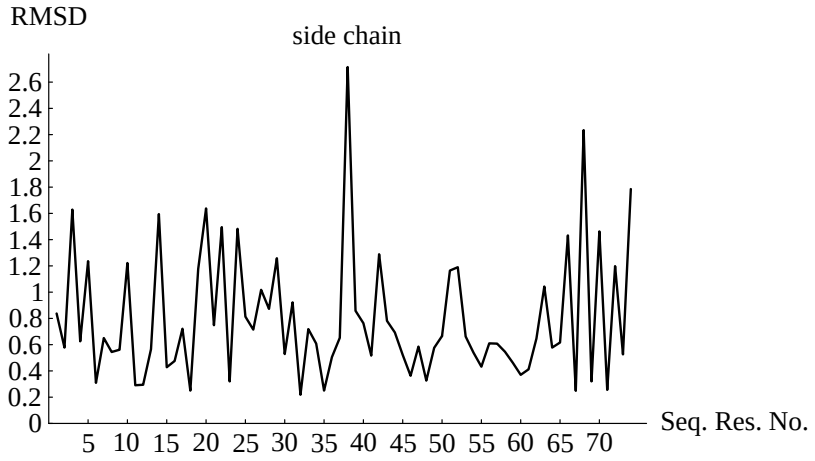

Figure 6.2: Rmsd vs. residue number.

```
                    (name %G* or name %D* or name %E* or name %H*))
      center (head (resid $resid and name cg) )
      axis (head (resid $resid and name cg)
            tail (resid $resid and name cb)) 180
  end

  coor rms end
  evaluate ($newrms=$result)

  if ($newrms > $oldrms) then
    coor rotate sele= (resid $resid and
                      (name %G* or name %D* or name %E* or name %H*))
      center (head (resid $resid and name cg) )
      axis (head (resid $resid and name cg)
            tail (resid $resid and name cb)) 180
    end
  end if

end loop rotr

for $id in id ( (resn ASP) and name ca ) loop rotr

  vector show element (resid) ( id $id )
  evaluate ($resid = $result)

  coor rms end
  evaluate ($oldrms=$result)

  coor rotate sele= (resid $resid and (name %G* or name %D*))
      center (head (resid $resid and name cb) )
      axis (head (resid $resid and name cb)
            tail (resid $resid and name ca)) 180
  end


  coor rms end
  evaluate ($newrms=$result)

  if ($newrms > $oldrms) then
    coor rotate sele= (resid $resid and (name %G* or name %D*))
      center (head (resid $resid and name cb) )
      axis (head (resid $resid and name cb)
            tail (resid $resid and name ca)) 180
```

```
    end
  end if

end loop rotr


for $id in id ( (resn ARG) and name ca ) loop rotr

  vector show element (resid) ( id $id )
  evaluate ($resid = $result)

  coor rms end
  evaluate ($oldrms=$result)

  coor rotate sele= (resid $resid and (name %Z* or name %H*))
       center (head (resid $resid and name cz) )
       axis (head (resid $resid and name cz)
             tail (resid $resid and name ne)) 180
  end


  coor rms end
  evaluate ($newrms=$result)

  if ($newrms > $oldrms) then
    coor rotate sele= (resid $resid and (name %Z* or name %H*))
       center (head (resid $resid and name cz) )
       axis (head (resid $resid and name cz)
             tail (resid $resid and name ne)) 180
    end
  end if

end loop rotr

set message on echo on end
```

This example file is specific for proteins but it could easily be modified for other biopolymers.

6.4 Distance Matrix Analysis

In the next example, a distance matrix is produced among the mass-weighted centroids of all residues of the protein.

```
remarks file geomanal/distance_plot.inp -- Makes a matrix of all
remarks residue centroid to residue centroid distances.  The
remarks matrix can be interpreted by Mathematica.

{===>}                                                {*Read structure file.*}
structure @../generate/generate.psf end

{===>}                                                  {*Read coordinates.*}
coordinates @../xtalrefine/slowcool.pdb

{===>}
vector ident (store9) ( tag )              {*This selects all residues to  *}
                                           {*be analyzed.   Change it      *}
                                           {*to analyze portions,          *}
                                           {*e.g.,  ( tag and segid "A" ). *}
                                           {*"tag" assigns one unique atom *}
                                           {*per residue (see <selection>).*}

parameter
{===>}
   @TOPPAR:parhcsdx.pro                {*Read empirical energy parameters.*}
```

```
end

set message off echo off end                      {*This reduces the output.*}
set abort off end


{====>}
set display distance_plot.matrix end               {*Filename of the matrix.*}

display  matrix={


evaluate ($count1=0)
for $id1 in id ( tag ) loop id1
   if ($count1=0) then
      display  {
   else
      display ,{
   end if
   evaluate ($count1=$count1+1)

   evaluate ($count2=0)
   for $id2 in id ( tag ) loop id2

  {*The following statement computes the distance between the mass-weighted*}
  {*centroids of the two residues.                                          *}
      pick bond
        (byresidue ( id $id1)) (byresidue ( id $id2)) geometry

      evaluate ($distance=$result)

      if ($count2=0) then
         display  $distance
      else
         display , $distance
      end if
      evaluate ($count2=$count2+1)

   end loop id2
   display }
end loop id1

display };


stop
```

A file "distance.matrix" is produced, which can be interpreted by Mathematica using the following statements:

```
             (* file geomanal/distance_plot.math                          *)
             (* mathematica file -- Interprets the 2-D matrices written by *)
             (* the distance_plot.inp example file                         *)
Off[General::spell1];
$DefaultFont={"Times-Roman",13};

(*===>*)
  << distance_plot.matrix;          (* Specify the file name of the matrix *)
                                    (* written by X-PLOR.                  *)


densplot =
  ListDensityPlot[
    matrix,
    PlotRange -> {0,12},             (* Plot distances between 0 and 12 A. *)
    Mesh -> False ]
```

The resulting density plot of the distance matrix is shown in Fig. 6.3.

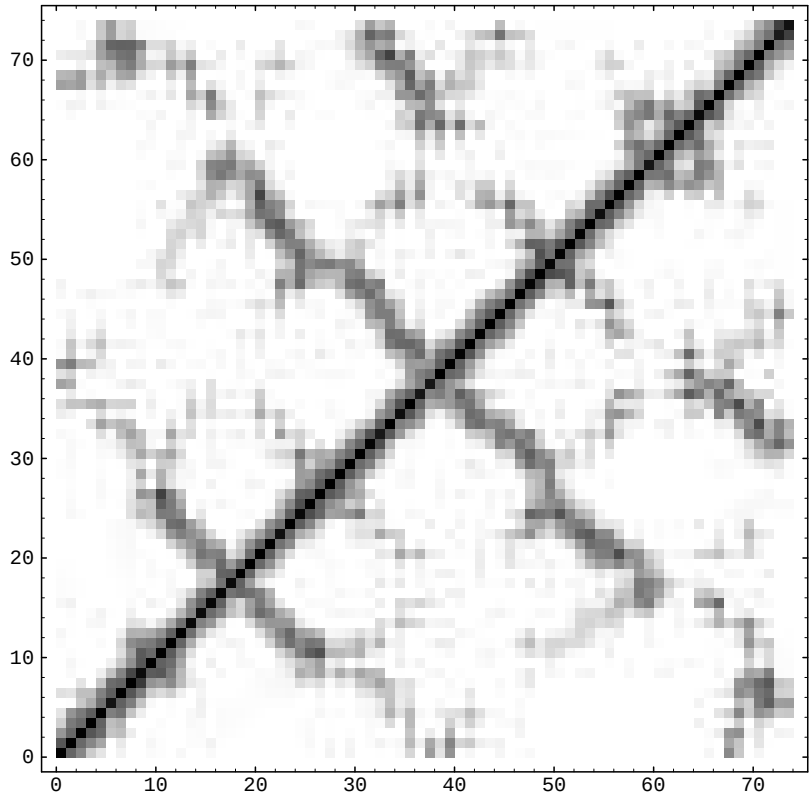

Figure 6.3: Distance matrix.

The pick bond statement is used to produce the distance matrix. It can be replaced by any other statement or series of statements in X-PLOR. For example, one could compute the interaction energy between the pair of residues using the constraints interaction and the energy statements (see Section 4.7). In the next example, a distance difference matrix is produced between two coordinate sets:

```
1  remarks file geomanal/distance_diff_plot.inp -- Makes a matrix of all
2  remarks residue centroid to residue centroid distance differences
3  remarks between the coordinates in the main set and in the comparison
4  remarks coordinate set.  The matrix can be interpreted by Mathematica.
5
6  {===>}                                              {*Read structure file.*}
7  structure @../generate/generate.psf end
8
9  {===>}                                                 {*Read coordinates.*}
10 coordinates @../xtalrefine/slowcool.pdb
11 coordinates disp=comp @../generate/generate.pdb
12
13 {===>}
14 vector ident (store9) ( tag )              {*This selects all residues to  *}
15                                            {*be analyzed.   Change it      *}
16                                            {*to analyze portions,          *}
17                                            {*e.g., ( tag and segid "A" ).  *}
18                                            {*"tag" assigns one unique atom *}
19                                            {*per residue (see <selection>).*}
```

```
20
21 parameter
22 {===>}
23    @TOPPAR:parhcsdx.pro                 {*Read empirical energy parameters.*}
24 end
25
26 set message off echo off end                      {*This reduces the output.*}
27 set abort off end
28
29
30 {====>}
31 set display distance_diff_plot.matrix end         {*Filename of the matrix.*}
32
33 display  matrix={
34
35
36 evaluate ($count1=0)
37 for $id1 in id ( tag ) loop id1
38    if ($count1=0) then
39       display  {
40    else
41       display ,{
42    end if
43    evaluate ($count1=$count1+1)
44
45    evaluate ($count2=0)
46    for $id2 in id ( tag ) loop id2
47
48   {*The following statement computes the distance between the mass-weighted*}
49   {*centroids of the two residues.                                          *}
50       pick bond
51         (byresidue ( id $id1)) (byresidue ( id $id2)) geometry
52       evaluate ($distance1=$result)
53       coor swap end
54       pick bond
55         (byresidue ( id $id1)) (byresidue ( id $id2)) geometry
56       evaluate ($distance2=$result)
57       evaluate ($diff=$distance1-$distance2)
58       coor swap end
59
60       if ($count2=0) then
61          display  $diff
62       else
63          display , $diff
64       end if
65       evaluate ($count2=$count2+1)
66
67    end loop id2
68    display }
69 end loop id1
70
71 display };
72
73
74 stop
```

After interpreting the file "distance_diff.matrix" with Mathematica, one obtains the plot shown in Fig. 6.4.

6.5 Building Hydrogen Positions

The hbuild statement tries to build the position of any selected hydrogen based on the position of the heavy atom antecedents (Brünger and Karplus 1988). It works in a general way and can be used with any empirical force

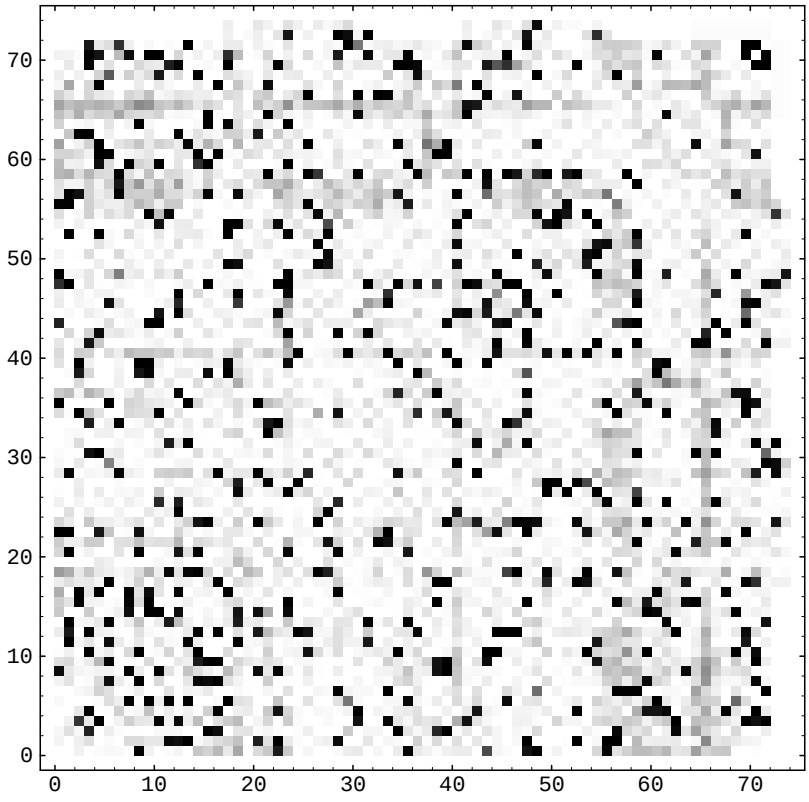

Figure 6.4: Difference distance matrix.

field. It performs local energy minimization in cases where the placement of the hydrogens is not unique. Waters close to the macromolecule are placed first, followed by waters that are farther away. Several iterations can be carried out to reach self-consistency. The minimization uses the energy function (Eq. 4.1) except that the specification of constraints interaction statements (Section 4.7) are ignored.

It is recommended to perform minimization of all hydrogen positions using the minimize Powell statement (Section 9.1) .

The hydrogen building facility does not use the improper or dihedral information in the topology file (Section 3.1.1). The chirality of methyl and methylene groups is determined by the order of four substituents to the central carbon atom as specified in the molecular structure. By changing the order in which the atoms are defined in the residue statement (Section 3.1.1), the chirality of the center is changed.

Hydrogens are recognized by X-PLOR by their mass. All atoms with a mass less than 3.5 amu are considered hydrogens. Certain simulated annealing protocols (e.g., Section 20.3.3) make use of artifically increased masses. If hydrogen building is employed in these protocols, the hydrogen building routine will return an error statement. In this case the user has to reset the

mass of the hydrogen atoms to their physical values.

To save CPU time, it is suggested to carry out the hydrogen building without nonbonded energy terms, followed by energy minimization of the hydrogen positions using the minimize powell statement.

6.5.1 Syntax

HBUIld { <**hbuild-statement**> } **END** is invoked from the main level of X-PLOR. The END statement activates hydrogen building.

<**hbuild-statement**>:==

ACCEptor=<**selection**> selects atoms that should be perceived as acceptors for hydrogen bonds involving waters (default: all atoms that have an explicit ACCEptor assignment; see Section 3.1.1).

CUTWater=<**real**> specifies the cutoff value for acceptors involving hydrogen bonds to waters (default: 7.5 Å).

PHIStep=<**real**> specifies the step size in degrees for the dihedral angle search in cases where the placement of the hydrogens is not unique (default: 10°).

PRINt is a flag that provides information about the partial energies during the local minimizations.

SELEction=<**selection**> specifies a selection of atoms that should be built. Only hydrogens may be specified; hydrogens are defined in X-PLOR by their mass being less than 3.5 amu. An error statement will be returned if any of the selected atoms has a larger mass (default: (HYDRogen AND NOT KNOWn)).

6.5.2 Requirements

The molecular structure, heavy atom coordinates, and parameters have to be defined. All energy terms are used that are specified by the flag statement. All databases are preserved.

6.5.3 Example

In the following example, all hydrogens of the molecule are built. Initially, the nonbonded energy terms are excluded to reduce computing time. The hydrogen positions are then minimized using the nonbonded energy terms.

```
flags exclude vdw elec end
hbuild
   selection=( hydrogen )
   phistep=45.0
end
flags include vdw elec end
constraints fix=( not hydrogen ) end
minimize powell
```

```
 9    nstep=40
10 end
```

7 Coordinate Restraints

This section deals with restraining certain Cartesian or internal coordinates. Restraints that are discussed here can be imposed as atomic rms differences to another coordinate set or as dihedral angle differences. (Non-crystallographic symmetry restraints are discussed in Chapter 16. Distance restraints are discussed in Chapter 18.)

7.1 Harmonic Coordinate Restraints

7.1.1 Point Restraints

X-PLOR provides a means to restrain the main coordinate set (atom properties x,y,z) to the reference coordinate set (atom properties refx, refy, refz). The restraints are defined as

$$E_{HARM} = \sum_{atoms} h_i (r_i - r_i^{ref})^e \tag{7.1}$$

where the sum extends over all atoms, h_i is individual weights, r_i is the atomic coordinates of the main set, r_i^{ref} is the atomic coordinates of the reference set, and e is an exponent. The individual weights h_i correspond to the atom property HARM and can be assigned using the vector statement (Section 2.16); the physical dimension of HARM is kcal mole^{-1} Å^e. By default, h_i is zero. The reference coordinate set can be assigned using the vector statement or by using the DISPosition option of the coordinate statement. The exponent e is set by using the restraints harmonic statement.

7.1.2 Plane Restraints

It is possibile to apply restraints to an individual atom based on its distance from a plane defined by the normal vector $\vec{n}$ and the atom's reference coordinate (atom properties refx, refy, refz). An atom for which plane restraints are defined experiences restraints only in the direction of $\vec{n}$. Plane restraints are defined as

$$E_{HARM} = \sum_{\text{atoms with } h_i<0} (-h_i) \left[\frac{\vec{n}}{|\vec{n}|} \cdot \left(\vec{r_i} - \vec{r_i^{ref}} \right) \right]^e \tag{7.2}$$

where the sum extends over all atoms with negative individual weights h_i. The symbols in Eq. 7.2 correspond to those in Eq. 7.1. By default, plane restraints are turned off. To enable plane restraints, a nonzero normal vector $\vec{n}$ has to be specified using the restraints harmonic statement and assigning a negative weight h_i for the atom property HARM. Note that plane restraints are computed only for atoms with $h_i < 0$; otherwise point restraints (see Section 7.1.1) are applied, even if a nonzero normal vector is defined. This permits the simultaneous use of point and plane restraints, however, for disjunct sets of atoms. If $\vec{n} = \vec{0} = (0,0,0)$, plane restraints are disabled.

7.1.3 Syntax

restraints HARMonic { <**restraints-harmonic-statement**> } **END**
is invoked from the main level of X-PLOR. This statement automatically turns on the HARM energy flag (Section 4.5).

<**restraints-harmonic-statement**>:==

EXPOnent=<**integer**> specifies the exponent e (default: 2).

NORMal=<**vector**> specifies the normal vector $\vec{n}$. If $\vec{n} \neq (0,0,0)$, plane restraints are enabled (default: (0 0 0)).

NORMal= (0 0 0) disables plane restraints.

7.1.4 Requirements

The molecular structure and the main and reference coordinate sets have to be defined.

7.1.5 Example: Point Restraints

The following example shows how to restrain a subset (in this case the C^α atoms) of the coordinates in "file1" to the coordinates in "file2":

```
coordinates @file1
coordinates disposition=reference @file2
vector do ( harmonic=20.0 ) ( name ca )
vector do ( harmonic=0.0 )  ( not name ca )
restraints harmonic exponent=2 end
flags include harm end
```

This sequence of statements can be inserted anywhere after the molecular structure is defined. Note that the FLAGs statement is actually not needed, because the restraints statement has the HARM flag already turned on.

7.1.6 Example: Plane and Point Restraints

This example demonstrates the simultaneous use of plane and point restraints:

```
coordinates @file1
vector do (refx=x) (all)
vector do (refy=y) (all)
vector do (refz=z) (all)
vector do ( harmonic=0.0 )  ( all )
vector do ( harmonic=20.0 ) ( resid 1 and name ca )
vector do ( harmonic=-20.0 ) ( resid 10 and name ca )
restraints harmonic exponent=2 normal=(0 0 1) end
flags include harm end
dynamics Verlet nstep=1000 timestep=0.001 firsttemp=300 end
```

The C^{α} atom of residue 1 will be harmonically restrained to its initial position in "file1". However, the C^{α} atom of residue 10 will be restrained only in the z direction; its motions are unrestricted parallel to the x-y plane.

7.2 Dihedral Angle Restraints

This section shows how to restrain dihedral angles to particular values. The functional form of the effective energy E_{CDIH} is given by

$$E_{CDIH} = S \sum C \; well(modulo_{2\pi}(\phi - \phi_o), \Delta\phi)^{ed} \tag{7.3}$$

where the sum extends over all restrained dihedral angles and the square-well potential $well(a,b)$ is given by

$$well(a,b) = \begin{cases} a-b & \text{if } a > b \\ 0 & \text{if } -b < a < b \\ a+b & \text{if } a < -b \end{cases} \tag{7.4}$$

S is an overall weight factor. The constant C, the angle range $\Delta\phi$, the angle centroid ϕ_o, and the exponent ed are specified in the restraints dihedral statement. The dihedral angle can be defined by a set of four atoms independent of the molecular structure. In particular, the four atoms do not have to be connected to each other. Each atom is selected by a separate selection statement. The dihedral angle involving four atoms i,j,k,l is defined as the angle between two planes, where the first plane is made by atoms i,j,k and the other plane is made by atoms j,k,l.

7.2.1 Syntax

RESTraints DIHEdral { <**restraints-dihedral-statement**> } **END** is invoked from the main level of X-PLOR. Invocation automatically turns on the CDIH energy flag (Section 4.5).

<**restraints-dihedral-statement**>:==

ASSIgn <**selection**> <**selection**> <**selection**> <**selection**> <**real**> <**real**> <**real**> <**integer**> adds a new dihedral angle to the restraints-dihedral database. The four selections have to be unique; i.e., each selection must select exactly one atom. The selections determine the four atoms that specify the dihedral angle. The first <real> number specifies the energy constant (C) in kcal mole^{-1} rad^{-2}, the second <real> number specifies the angle in degrees to which the dihedral angle is restrained (ϕ_o), the third <real> number specifies the range around the restrained angle ($\Delta\phi$), and the <integer> number specifies the exponent (ed).

NASSign=<**integer**> is a required parameter that specifies the maximum expected number of assignments. It has to be greater than or equal to the actual number (default: 400).

RESEt erases the restraints-dihedral database.

SCALe specifies the overall weight factor S.

7.2.2 Requirements

The molecular structure and atomic coordinates have to be defined. The dihedral angle restraints are independent of the parameter database. The atom selections are fragile (Section 2.15).

7.2.3 Example

This example shows how to set up dihedral angle restraints between selected atoms.

```
1  restraints dihedral
2         nassign=300
3         assign ( resid 1 and name ca )
4                ( resid 10 and name cb )
5                ( resid 4 and name n )
6                ( resid 8 and name sg ) 20.0  55.0 0.0 2
7
8         assgin
9                ( resid 3 and name hg )
10               ( resid 5 and name o )
11               ( resid 2 and name cb )
12               ( resid 1 and name cg ) 20.0 170.0 0.0 2
13       scale=1.0
14 end
15 flags include cdih end
```

7.3 Planarity Restraints

Although X-PLOR provides improper angles to maintain planarity of groups of atoms, the specification of these energy terms is sometimes cumbersome, especially when many atoms are involved. The restraints planarity statement

allows one to define an effective energy term E_{PLAN} that penalizes out-of-plane conformations of the selected atoms. This feature should be used for relatively few groups, as it involves a certain amount of overhead. The planarity restraints energy term is defined as

$$E_{PLAN} = \sum_{g \in groups} w_{plan} \sum_{i \in g} g_i^2 \tag{7.5}$$

where the first sum is carried out over all defined groups of planar atoms, the second sum is carried out for all atoms i within each group, and g_i is the orthogonal distance of i from the least-squares plane defined by all atoms of the group (Schomaker et al. 1959). Note that the energy term must be activated through the flags statement (Section 4.5).

Upon evaluation of the planarity restraints energy term the following symbols are declared: \$PLANX, \$PLANY, \$PLANZ. They represent the unit vector that is normal to the plane defined by the last group of atoms.

7.3.1 Syntax

RESTraints PLANar { <**restraints-planar-statement**> } **END** is invoked from the main level of X-PLOR.

<**restraints-planar-statement**>:==

GROUp { <**restraints-plane-group-statement**> } **END** adds a new group to the planar restraints database. More than three atoms per group need to be defined (default: (NOT ALL)).

INITialize erases the current planar restraints database.

<**restraints-plane-group-statement**>:==

SELEction=<**selection**> defines the group of atoms that are restrained to form a plane (default: (NOT ALL)).

WEIGht=<**real**> specifies a weight w_{plan} for the planarity restraint in kcal mole^{-1} Å^{-2} (default: 300.0 kcal mole^{-1} Å^{-2}).

7.3.2 Requirements

The molecular structure and atomic coordinates have to be defined. The atom selection is fragile (Section 2.15).

7.3.3 Example

In the following example, planarity restraints are defined that restrain one group consisting of four DNA bases. This will ensure that the four bases form a plane. Additional planarity restraints could be defined by specifying more group statements.

```
restraints plane
  group

   selection=(
          (not( name P   or name O1P or name O2P or name O5' or name H5T or
                name H1' or name H2' or name H3' or name H4' or name H5' or
                name H2'''' or name H5'''' or
                name O3' {or name C1'} or name C2' or name C3' or name C4' or
                name C5' or name O4' or name H3T) and residue 1 )
        or  (not( name P   or name O1P or name O2P or name O5' or name H5T or
                name H1' or name H2' or name H3' or name H4' or name H5' or
                name H2'''' or name H5'''' or
                name O3' {or name C1'} or name C2' or name C3' or name C4' or
                name C5' or name O4' or name H3T) and residue 6 )
         or (not( name P   or name O1P or name O2P or name O5' or name H5T or
                name H1' or name H2' or name H3' or name H4' or name H5' or
                name H2'''' or name H5'''' or
                name O3' {or name C1'} or name C2' or name C3' or name C4' or
                name C5' or name O4' or name H3T) and residue 10 )
         or (not( name P   or name O1P or name O2P or name O5' or name H5T or
                name H1' or name H2' or name H3' or name H4' or name H5' or
                name H2'''' or name H5'''' or
                name O3' {or name C1'} or name C2' or name C3' or name C4' or
                name C5' or name O4' or name H3T) and residue  15 )
    )

    weight=400.0
  end
  ?
end
                        {* The planarity energy term needs to be turned on.*}
flags include plan end
```

8 Coordinate Constraints

This section deals with constraining certain coordinates. Constraints can be imposed as rigid distance requirements using the SHAKE algorithm or as fixed atomic positions.

8.1 Fixing Atomic Positions

Atomic positions can be fixed during minimization or molecular dynamics.

8.1.1 Syntax

CONStraints FIX <constraints-fix-statement> END is invoked from the main level of X-PLOR.

<constraints-fix-statement>:==

> **<selection>** selects atoms that should be fixed during minimization or molecular dynamics (default: none). The selection of fixed atoms remains active until a new constraints fix statement is issued.

8.1.2 Requirements

The molecular structure has to be defined. The atom selection is partially fragile (Section 2.15); i.e., the information about fixed atoms is mapped when one is modifying the molecular structure.

8.1.3 Example

The following example fixes C^{α} carbon atoms:

```
1 constraints fix=( name ca ) end
```

8.2 Fixing Distances

The SHAKE method (Ryckaert, Ciccotti, and Berendsen 1977) constrains distances between atoms to certain reference distances. In this way, one can

constrain bond lengths or angles by constraining the distance between the first and the third atoms of the angle (ijk). The shake statement is used to set up the database for the SHAKE constraints. SHAKE constraints are active during minimization or molecular dynamics, or they can be explicitly imposed by the SHAKE option of the coordinate statement. At present, SHAKE constraints cannot be used during rigid-body minimization.

8.2.1 Syntax

SHAKe { <**shake-statement**> } **END** is invoked from the main level of X-PLOR.

<**shake-statement**>:==

ANGLe <**selection**> <**selection**> <**selection**> adds new SHAKE constraints to the SHAKE database. The program goes through the list of bond angles of the molecular structure, trying to match them against the triple atom selection. If the first and third atoms of the angle match the first and the third selections or vice versa, and if the second atom matches the second selection, a SHAKE constraint of the distance between the first and third atoms is added to the database. For parameter-based SHAKE angle constraints (REFErence=PARAmeter), type-based parameters will be retrieved for both the bond angle and the bond lengths. Thus, the type-based parameter database needs to contain the corresponding bond and bond angle parameters. The atom-based parameter database has no influence on this statement (default: none).

BOND <**selection**> <**selection**> adds new SHAKE constraints to the SHAKE database. The program goes through the list of covalent bonds of the molecular structure, trying to match them against the double atom selection. If one atom of the bond matches the first selection and the other atom matches the second selection, a SHAKE constraint of this bond distance is added to the database. For parameter-based SHAKE bond constraints (REFErence=PARAmeter), the atom- and type-based parameter databases are consulted in order to retrieve the required bond length (default: none).

MOLEcule <**selection**> adds new SHAKE constraints to the SHAKE database. The program goes through all interatom distances of the selected atoms that belong to the same nonbonded group (see Section 3.1.1) and adds to the database a SHAKE constraint of the interatom distance. Normally, this statement is used to SHAKE small molecules such as waters.

MXITerations=<**integer**> specifies the maximum number of SHAKE iterations that are allowed before the SHAKE procedure exits with an error condition (default: 500).

NCONstraints=<integer> is a parameter that allocates space for SHAKE constraints. The specified <integer> should be greater than or equal to the actual number of SHAKE constraints (default: 4000).

REFErence= COORdinates | PARAmeters determines whether the reference distances for the SHAKE constraints will come from the main coordinate set or from the parameters. This statement has to be specified before the MOLEcule, BOND, or ANGLe statements (default: COORdinate).

RESEt erases the current SHAKE database.

TOLErance=<real> specifies the point at which the SHAKE iterations are terminated (default: 1.0e-05). The tolerance is satisfied if the deviation of the constrained distances is less than the tolerance times the actual distance.

8.2.2 Requirements

The molecular structure, atomic coordinates, and parameters have to be defined. Care should be taken that the SHAKE constraints do not involve any fixed atoms (Section 8.1). The atom selections are fragile (Section 2.15).

8.2.3 Example

The following example shows how to SHAKE all bond lengths between hydrogens and heavy atoms, and the geometry of all TIP3 water molecules.

```
1 shake
2    reference=parameters  molecule ( resname TIP3 )
3    reference=coordinates  bonds ( hydrogen and not resname TIP3 )    ( all )
4    tolerance=1.0e-06
5    mxiterations=500
6 end
```

9 Energy Minimization

X-PLOR provides two options to minimize the total energy E_{TOTAL} (Chapter 4). The conjugate gradient minimizer uses the Cartesian coordinates as variables. The rigid-body minimizer uses the rotational and translational degrees of freedom for specified rigid groups of the molecule.

9.1 Conjugate Gradient Minimization

X-PLOR uses the conjugate gradient method by Powell (1977). The minimization is started from the atom properties X,Y,Z, and the minimized coordinates are returned in X,Y,Z. Only the coordinates of free atoms (i.e., not fixed atoms) will be modified during the minimization (cf. Section 8.1). SHAKE constraints are possible (cf. Section 8.2). Upon completion of the last energy calculation, symbols are declared that contain the computed energy terms. The name of the symbols is given by $<energy-term> (see Section 4.5). The overall energy (Eq. 4.1) is stored in the symbol $ENER; the rms gradient is stored in $GRAD. The value of the second energy function (Eq. 4.26) is returned in the symbol $PERT.

9.1.1 Syntax

MINImize POWEll { <**minimize-powell-statement**> } **END** is invoked from the main level of X-PLOR.

<**minimize-powell-statement**> **:==**

DROP=<real> designates the expected initial drop in energy (default: 0.001). Experience has shown that values between 10 and 100 work best.

NPRInt=<integer> is the frequency of the energy printout (default: 1).

NSTEp=<integer> is the maximum number of minimization cycles (default: 500).

TOLGradient=<real> stops minimization if the norm of the gradient of E_{TOTAL} is smaller than the specified number (default: 0.0001).

9.1.2 Requirements

The molecular structure, parameters, and atomic coordinates have to be defined.

9.1.3 Example

```
minimize powell
   nstep=40
   drop=10
end
```

9.2 Rigid-Body Minimization

The rigid-body method minimizes the six rotational and translational degrees of freedom for each specified group of atoms; i.e., the groups of atoms are treated as rigid bodies. The complete energy function E_{TOTAL} is used. The rotational parameters are the three Eulerian angles $\theta_1, \theta_2, \theta_3$ for a rotation around the geometric centers of the rigid group. The three translational parameters are x,y,z in the Å frame. Parts of the molecule that are not specified in any GROUp statement will remain fixed. The constraints fix statement (Section 8.1) has no influence on rigid-body minimization (except for the default group selection). Upon completion of the last energy calculation, symbols are declared that contain the computed energy terms. The name of the symbols is given by $<energy-term> (see Section 4.5). The overall energy (Eq. 4.1) is stored in the symbol $ENER; the rms gradient is stored in $GRAD. The value of the second energy function (Eq. 4.26) is returned in the symbol $PERT.

9.2.1 Syntax

MINImize RIGId { <**minimize-rigid-statement**> } **END** is invoked from the main level of X-PLOR.

<**minimize-rigid-statement**> **:==**

DROP=<**real**> designates the expected initial drop in energy (default: 1).

GROUp=<**selection**> selects atoms that will be forming a rigid group. Multiple specification of GROUps will define multiple rigid groups. The group selections have to be disjoint (default: one group consisting of all nonfixed atoms). The default is overwritten as soon as a GROUp statement is issued.

NPRInt=<**integer**> is the frequency of the energy printout (default: 1).

NSTEp=<**integer**> is the maximum number of minimization cycles (default: 50).

TOLErance=<real> stops minimization if the norm of the gradient of the energy function E_{TOTAL} is smaller than the specified number (default: 0.01).

TRANslation=<logical> determines if the translational components of the last group are refined. (default: TRUE).

Requirements

The molecular structure, parameters, and atomic coordinates have to be defined.

9.2.2 Example

```
minimize rigid
  nstep=60   drop=10.
  group=( residue 1:6 )    group=( residue 7:19 )
  group=( residue 20:22 )  group=( residue 23:30 )
  group=( residue 31:35 )  group=( residue 36:41 )
  group=( residue 42:46 )
end
```

10 Molecular Dynamics

Molecular dynamics consists of solving Newton's equations of motion

$$m_i \frac{d^2 x_i}{dt^2}(t) = -\nabla_{x_i} E_{TOTAL} \tag{10.1}$$

where the index i runs through all free (i.e., not fixed) atoms and the gradient $\nabla_{x_i} E_{TOTAL}$ is derived from the X-PLOR energy function (Chapter 4). Presently, X-PLOR provides the option to solve Eq. 10.1 in Cartesian coordinate space (Section 10.1) or rigid-body coordinate space (Section 10.2). Molecular dynamics simulations can be stored as trajectory files. Input format, output format, manipulations, and analysis of trajectory files are described in Section 10.

10.1 Cartesian Coordinate Space

Normally, all atoms are free to move during molecular dynamics. However, atoms can be fixed by using the constraints fix statement (Section 8.1). The atomic masses m_i are defined through the topology statement (Section 3.1.1). The initial atomic coordinates are taken from the main coordinate set (atom properties X,Y,Z). After completion of the desired molecular dynamics steps, the main coordinate set contains the coordinates of the last step. The initial velocities are determined by various velocity assignment options or can be set explicitly by using the vector statement for the atom properties VX, VY, VZ. After completion of the molecular dynamics run, the velocities of the last step are stored in the atom properties VX, VY, VZ. The SHAKE method (Section 8.2) can be used during molecular dynamics to constrain specified distances, such as bond lengths.

After completion of the molecular dynamics calculation, the partial energy terms for the last molecular dynamics step are stored in the appropriate symbols. The name of the symbols is given by $<energy-term> (see Section 4.5). The overall energy (Eq. 4.1) is stored in the symbol $ENER; the rms gradient is stored in $GRAD. The value of the second energy function (Eq. 4.26) is returned in the symbol $PERT. In addition, the following symbols are declared: $TEMP, $TOTE, and $TOTK, which are respectively the temperature, total energy, and kinetic energy. If the IPRFRQ frequency is

nonzero, the average values of the energy terms and the temperature are stored in symbols whose names are preceeded by "AVE_"; e.g., the average bond energy is stored in the symbol $AVE_BOND. These averages are computed over the whole molecular dynamics run.

10.1.1 Simple Langevin Dynamics

X-PLOR provides the possibility of carrying out Langevin dynamics. The simple Langevin equation has the form

$$m_i \frac{d^2 x_i}{dt^2}(t) = -grad_{x_i} E + f_i(t) - m_i b_i \frac{dx_i}{dt}(t) \tag{10.2}$$

where the random force $f_i(t)$ is derived from a Gaussian distribution with the properties

$$< f_i(t) >= 0 \tag{10.3}$$

$$< f_i(t) f_i(0) >= 2 k_B T_o b_i m_i \delta(t) \tag{10.4}$$

where k_B is Boltzmann's constant and T_o is the temperature specified by TBATh. For Langevin heatbath applications, the total energy E_{TOTAL} can be augmented by a harmonic energy term for all atoms with a nonzero friction coefficient b_i. The setup of the harmonic energy is described in Section 7.1. The friction coefficient b_i is defined by assigning the FBETa atom property (see Section 2.16). Langevin dynamics is used for all atoms that have nonzero friction coefficients b_i and that are outside a sphere with radius RBUF (default 0) around the origin ORIGIn. This check is carried out periodically (ILBFrq parameter). By default the atom property FBETa is zero; i.e., all atoms are treated as normal particles.

10.1.2 Velocity Assignment

The initial velocities can be assigned to a Maxwellian distribution (IASVel = MAXWell)

$$(\frac{m_i}{2\pi k_B T_I})^{3/2} exp(\frac{-3 m_i \delta^2}{2 k_B T_I}) \tag{10.5}$$

where δ is a random number between 0 and 1. T_I is the initial temperature specified through the parameter FIRStemperature. Alternatively, the velocities can be assigned to a uniform distribution with width $T_I k_B / m_i$ (IASVel=UNIForm). As a third possibility, the initial velocities can be taken from the atom properties VX, VY, VZ and indicated by IASVel= CURRent.

10.1.3 Temperature Control

X-PLOR has three options to maintain the temperature of the simulation.

Velocity Rescaling

The velocity-rescaling option (IEQFrq not zero) periodically rescales the velocities. It uses the equations

$$v_i^{new} = \begin{cases} v_i^{old}\sqrt{3nk_BT_o/ < \sum_i mv_i^{old}(t)^2 >} & \text{if ISCVel=0} \\ v_i^{old}\sqrt{3k_BT_o/ < mv_i^{old}(t)^2 >} & \text{if ISCVel=1} \end{cases} \tag{10.6}$$

where n is the number of free atoms in the system, k_B is Boltzmann's constant, T_o is the temperature in K specified by the FINAltemperature parameter, $V_i^{old}(t)$ denotes the velocity of atom i at time t, and angle brackets enclose a trajectory average over the time intervals between the rescalings.

Langevin Dynamics

Langevin dynamics is turned on by a nonzero FBETA (friction) array, a nonzero ILBFrq parameter, and an appropriate choice of RBUF. Langevin dynamics represents a coupling of the system to a heatbath and thus provides a means to control and maintain temperatures. A harmonic energy term can be used to restrain the Langevin particles and provide a rough approximation of a potential of mean force.

Temperature Coupling

This option is turned on by TCOUpling. This method was described by Berendsen et al. (1984). It is essentially a Langevin-type method with zero random forces and a scaled friction coefficient. The friction coefficient is computed by the program from the equation

$$b_i = b_i^I(T_o/T - 1) \tag{10.7}$$

where b_i^I is given by the FBETa atom property. Note that FBETa is by default zero; i.e., FBETa has to be specified in order to use TCOUpling. The target temperature T_o is specified by the TBATh parameter. $T = \sum_i m_i \frac{dx_i}{dt}^2/(n_{deg}k_B)$ is the temperature of the system at the particular step where n_{deg} is the number of degrees of freedom of the system. Thus, a positive fricition coefficient is applied if the system is too hot, a negative friction coefficient is applied if the system is too cold, and no friction is applied if T matches T_o.

10.1.4 Finite Difference Approximation

X-PLOR makes use of a third-order finite difference approximation in dt (Brünger, Brooks, and Karplus 1984). First, the initial coordinates ${x_i}^0$ are subjected to the SHAKE method. Then the system gets the initial velocities

${v_i}^0$. Next, the program prints the energy of the initial coordinates. A two-step method is used to obtain the coordinates ${x_i}^1$:

$$ {x_i}^1 = {x_i}^0 + {v_i}^0 \Delta t - \nabla_{x_o}(E_{TOTAL}) \frac{\Delta t^2}{2m_i} \tag{10.8} $$

IF SHAKE constraints are present, the SHAKE method is applied to ${x_i}^1$ with respect to ${x_i}^0$.

Iteration from step n to step $n+1$ causes $(x_i^{n-1}, x_i^n) \longrightarrow (x_i^{n+1}, v_i^n)$. The algorithm computes the forces $F_i^n = -\nabla_{x_i^n} E + f_i(t)$. The algorithm then computes

$$ x_i^{n+1} = [1 + \frac{b_i \Delta t}{2}]^{-1} [2x_i^n - x_i^{n-1} + F_i^n \frac{\Delta t^2}{m_i} + x_i^{n-1}(\frac{b_i \Delta t}{2})] \tag{10.9} $$

If required, the SHAKE method is applied to x_i^{n+1} with x_i^n as the reference set. Finally, the velocities at this step are computed:

$$ v_n = (\frac{1}{2\Delta t})(x_{n+1} - x_{n-1}) \tag{10.10} $$

(The velocities do not enter the equations to compute the trajectory x_i^n.) In case of zero friction coefficients b_i, this algorithm reduces to the three-step Verlet method (Verlet 1967).

10.1.5 Dynamics Restarts

In order to break up the dynamics stages into smaller jobs or to save the results in case of a fatal system crash, it is possible to write restart files periodically during the execution of the dynamics job. The restart option is turned on by inserting the following statement anywhere into the dynamics Verlet statement:

```
1 isvfrq=<integer>  save=<filename>
```

where ISVFRQ specifies the number of dynamics steps between updates of the restart file. Each time the restart file is updated, a new version of it is created.

To restart using an existing restart file, the following statement should be inserted immediately after the DYNAmics VERLet keywords:

```
1 restart=<filename>.
```

The number of steps after a restart should be reduced, depending on how many steps were carried out before the restart. The recovery after a restart is complete; i.e., the trajectory is identical to one produced by a job without restarts. However, the restart file contains information only about the coordinates of the steps $n-1$, n, and $n+1$. All other structural information and databases have to be read again before executing the dynamics.

10.1.6 Syntax of the Dynamics Verlet Statement

The dynamics Verlet statement sets up various parameters for molecular dynamics as well as Langevin dynamics and then executes the specified number of steps.

DYNAmics VERLet { <**dynamics-Verlet-statement**> } **END** is invoked from the main level of X-PLOR.

<**dynamics-Verlet-statement**> **:==**

ASCIi=<logical> writes a formatted (ASCII) coordinate trajectory file if ASCIi is TRUE; if ASCIi is FALSE, writes an unformatted (binary) file. In the former case, a scale factor s and an offset o are applied to the coordinates r (i.e., $r' = s * (r + o)$); the result is converted to an integer number and written to the ASCII file using the specified format. If a hexadecimal format is specified, care should be taken to ensure that the application of the scale factor s and the offset o results in positive coordinates during the whole course of the molecular dynamics simulation. Otherwise, FORTRAN format runtime errors will occur.

CSEL=<selection> selects atoms that are written to the coordinate trajectory file (default: all atoms except those fixed by the constraints fix statement).

FINAltemp=<real> determines the target temperature for the velocity rescaling option (see ISCVel) (default: 298K).

FIRSttemp=<real> assigns the temperature that determines the initial velocity (default: 0K).

FORMat=<string> provides FORTRAN format for the ASCIi= TRUE option. Only integer or hexadecimal formats are allowed. (default: 12Z.6, hexadecimal).

IASVelocity=MAXWell | UNIForm | CURRent determines the mode of initial velocity assignments (default: zero velocities).

IEQFrq=<integer> determines the frequency with which the velocities are rescaled (default: 0). It works in conjunction with FINAltemperature.

ILBFrq=<integer> updates Langevin boundary between normal and Langevin particles (default: 0). It works in conjunction with RBUF and ORIGin.

IPRFrq=<integer> prints energy statistics every IPRFrq steps if not equal to zero (default: 50).

ISCVel=<integer> determines the velocity rescale mode (default: 0).

ISVFrq=<integer> determines how often a restart file is written to disk (default: 0).

NPRInt=<integer> determines the frequency with which the energy is printed to standard output (default: 1, i.e., at every dynamics step).

NSAVC=<integer> determines the frequency with which the coordinates are written to the coordinate trajectory file (default: 0).

NSAVV=<integer> determines the frequency with which the velocities are written to the velocity trajectory file (default: 0).

NSTEp=<integer> determines the number of steps. In the case of a restart, the steps executed before writing the restart file are included in NSTEP (e.g., if NSTEP=3000, but 1000 steps were run before writing the restart file, the program will execute only 2000 steps) (default: 100).

NTRFrq=<integer> determines the frequency with which the center-of-mass motion is removed (default: 99999, i.e., just do once at the start).

OFFSet=<real> provides offset o for the ASCIi=TRUE option (default: 800).

ORIGin=<vector> is the origin of the sphere of the Langevin boundary (default: 0,0,0).

RBUF=<real> specifies the radius of the spherical boundary in a dynamics Verlet statement. Inside the boundary, the particles are "normal" molecular dynamics particles; outside they are propagated as Langevin particles (default: 0).

RESTart=filename reads restart information from the named file and restarts the dynamics (default: no restart).

SAVE=<filename> specifies the name of the file to which the program periodically writes the restart information (default: none).

SCALe=<real> provides a scale factor s for the ASCIi=TRUE option (default: 10000).

TBATh=<real> specifies the temperature of the heatbath or the target temperature of Berendsen's temperature-coupling method.

TCOUpling=<logical> is a logical flag that turns on Berendsen's temperature-coupling method. This flag works in conjunction with TBATh (default: FALSE).

TIMEstep=<real> assigns the time step Δt value for the finite-difference integration in psec (default: 0.001 psec).

TRAJectory=<file> provides the filename of the coordinate trajectory file (default: none).

VASCii=<logical> writes a formatted (ASCII) velocity trajectory file if VASCii is TRUE; if VASCii is FALSE, an unformatted (binary) velocity trajectory file is written. In the former case, a scale factor s and an offset o are applied to the velocities r (i.e., $r' = s_v * (r + o_v)$); the result is converted to an integer number and written to the ASCII file using

the specified format. If a hexadecimal format is specified, care should be taken to ensure that the application of the scale factor s_v and the offset o_v results in positive velocities during the whole course of the molecular dynamics simulation. Otherwise, FORTRAN format runtime errors will occur.

VELOcity=<filename> names the velocity trajectory file (default: none).

VFORmat=<string> is the FORTRAN format of the velocity trajectory file for the VASCii= TRUE option. Only integer or hexadecimal formats are allowed. (default: 12Z.6, hexadecimal).

VOFFset=<real> provides an offset o_v for the VASCii= TRUE option (default: 300).

VSCAle=<real> provides a scale factor s_v for the VASCii=TRUE option (default: 10000).

VSEL=<selection> selects atoms that are written to the velocity trajectory file (default: all atoms except those fixed by the constraints fix statement).

10.1.7 Requirements

The molecular structure, parameters, and atomic coordinates have to be defined.

10.1.8 Example: Run a Standard Molecular Dynamics Simulation

```
dynamics verlet
   nstep=1000
   timestep=0.001
   iasvel=maxwell
   firsttemperature=300
   nprint=25
end
```

10.1.9 Example: Run a Molecular Dynamics Simulation with Temperature Coupling

```
vector do (fbeta=100. ) ( all )
dynamics verlet
   nstep=1000
   timestep=0.001
   iasvel=maxwell
   firsttemperature=300
   nprint=25
   tcoupling=true
   tbath=300.
end
```

10.1.10 Example: Run a Slow-cooling Molecular Dynamics Simulation

The powerful X-PLOR shell language is able to produce a large number of possible cooling or heating schedules for simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983). The example below uses a slow-cooling annealing schedule.

```
set seed=432324368 end

vector do (vx=maxwell(4000.)) ( all )
vector do (vy=maxwell(4000.)) ( all )
vector do (vz=maxwell(4000.)) ( all )

vector do (fbeta=100.) ( all )

evaluate ($1=4000)
while ($1 > 300.0) loop main

   dynamics  verlet
     timestep=0.0005
     nstep=50
     iasvel=current
     nprint=5  iprfrq=0
     tcoupling=true tbath=$1
   end
   evaluate ($1=$1-25)
end loop main
```

10.1.11 Example: Run Langevin Dynamics

Note that Langevin dynamics is carried out for all selected particles. In the example below, RBUF is set to zero. This ensures that the Langevin dynamics is carried out for all particles. No updates of the Langevin boundary determination are necessary. Thus, the ILBFrq parameter is set to the number of steps (only one update at the start). A coordinate trajectory file is written every 5 steps.

```
vector do ( fbeta = 6.657235 ) ( segid bute and resid 1 )
set seed=71403814. end
dynamics langevin
    timestep=0.001 nstep=2500 ilbfrq=2500
    tbath=300. iasvel=maxwell  rbuf=0.0 origin=( 0. 0. 0. )
    trajectory=langtest1.dat nsavc=5
end
```

10.2 Rigid-Body Coordinate Space

Rigid-body dynamics solves Newton's equations of motion for rigid collections of atoms (Goldstein 1980). Atoms are collected into rigid groups, the motion of which is determined by summing the forces acting on all of a group's elements and integrating the rigid-body equations of motion. The atomic masses m_i are defined through the topology statement (Section 3.1.1).

The initial atomic coordinates are taken from the main coordinate set (atom properties X,Y,Z). After completion of a rigid-body dynamics run, the main coordinate set contains the coordinates of the last step. The initial velocities are taken from the atom properties VX,VY,VZ. They must be initialized outside the rigid-body dynamics statement. After completion of a rigid-body dynamics run, the velocities of the last step are stored in the atom properties VX, VY, VZ.

After completion of the molecular dynamics calculation, the partial energy terms for the last molecular dynamics step are stored in the appropriate symbols. The name of the symbols is given by $<energy-term> (see Section 4.5). The overall energy (Eq. 4.1) is stored in the symbol $ENER; the rms gradient is stored in $GRAD. The value of the second energy function (Eq. 4.26) is returned in the symbol $PERT. In addition, the following symbols are declared: $TEMP, $TOTE, and $TOTK, which are respectively the temperature, total energy, and kinetic energy.

X-PLOR's implementation of rigid-body molecular dynamics follows the algorithm described by Head-Gordon and Brooks (1991). The algorithm treats each group as a continuous mass distribution located at the center-of-mass position defined by

$$R_J = \frac{1}{M_J}\sum_{i\in J} m_i r_i \qquad M_J = \sum_{i\in J} m_i \tag{10.11}$$

and characterized by its inertia tensor $\mathbf{I_J}$, which has as elements

$$I_{\alpha\alpha} = \sum_{i\in J} m_i((r_i - R_J)^2 - (\alpha_i - \alpha_J)^2) \qquad \alpha \in (x,y,z) \tag{10.12}$$

$$I_{\alpha\beta} = \sum_{i\in J} -m_i(\alpha_i - \alpha_J)(\beta_i - \beta_J) \qquad \alpha,\beta \in (x,y,z) \tag{10.13}$$

Here the index J labels the rigid bodies, and the summation index i runs over all atoms comprising a particular group.

10.2.1 Initialization

Each inertia tensor is diagonalized by a rotational transformation to the body coordinate system:

$$\mathbf{I_J^P} = \mathbf{A_J^{-1}} \mathbf{I_J} \mathbf{A_J} \tag{10.14}$$

The transformation matrix $\mathbf{A_J}$ is used to initialize rotational variables such that $\mathbf{A_J}$=$(\mathbf{A_J^{-1}})^{\mathbf{t}}$. Values for the body-frame coordinates r_i^B of group elements are obtained by

$$r_i^B = \mathbf{A_J^t}(r_i - R_J) \tag{10.15}$$

The net force and torque acting on each body are determined by summing the force and torque acting on each of its constituent atoms. Center-of-mass

variables are initialized with a two-step process. The initial center-of-mass velocities are determined from the atom properties VX,VY,VZ:

$$V_J(t=0) = \frac{1}{M_J}\sum_{i\in J} m_i v_i \tag{10.16}$$

These velocities are then used to advance the center-of-mass coordinates

$$R_J(dt) = R_J(0) + V_J(0)dt + \frac{1}{M_J} f_j(0)(dt)^2 \tag{10.17}$$

The more stable Euler-Cayley parameters (also referred to as quaternions) are used as rotational variables instead of the Euler angles θ_1, θ_2, θ_3 (cf. Goldstein 1980). They are defined in Eq. 2.1. The quaternions are initialized using a first-order approximation to their equation of motion:

$$\dot{q} = \mathbf{Q}\omega' \tag{10.18}$$

where $\dot{q}$ is the four-vector ($\dot{q}_0$,$\dot{q}_1$, $\dot{q}_2$,$\dot{q}_3$), ω' is the four-vector (0,ω_x,ω_y,ω_z), and $\mathbf{Q}$ is the matrix that gives their time evolution:

$$\mathbf{Q} = \frac{1}{2}\begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix} \tag{10.19}$$

Thus one obtains

$$q(\frac{1}{2}dt) = q(0) + \frac{1}{2}\mathbf{Q}(0)\omega'(0)dt \tag{10.20}$$

The initial angular velocity ω follows directly from the initial angular momentum, which is determined by

$$J_J(0) = \sum_{i\in J}(r_i - R_J) \times p_i \tag{10.21}$$

where p_i is the momentum of the i^{th} atom of the rigid body. The initial half-step advanced angular momentum can be expressed as

$$J_J(\frac{1}{2}dt) = J_J(0) + \frac{1}{2}N(0) \tag{10.22}$$

and the first advancement of the center-of-mass coordinates can be written as

$$R_J(dt) = R_J(0) + V_J(0)dt + \frac{1}{2}f_J(0)\frac{(dt)^2}{M_J} \tag{10.23}$$

10.2.2 Iteration

After initialization, the center-of-mass variables are integrated using the three-step Verlet method:

$$R_J(t+dt) = -R_J(t-dt) + 2R_J(t) + f_J(t)\frac{(dt)^2}{M_J} \tag{10.24}$$

$$V_J(t) = (\frac{1}{2dt})(R_J(t+dt) - R_J(t-dt)) \tag{10.25}$$

Friction terms are embedded in the force f_J.

The integration of rotational variables uses a different central difference expansion. The algorithm begins with the advancement of the Euler-Cayley parameters by $\frac{1}{2}dt$, using the following four steps:

$$J(t) = J(t-\frac{1}{2}dt) + \frac{1}{2}N(t) \tag{10.26}$$

$$J^B(t) = \mathbf{A}(t)J(t) \tag{10.27}$$

$$\omega_\alpha^B = J_\alpha^B(t)/I_\alpha \qquad \alpha \in (x,y,z) \tag{10.28}$$

$$q(t+\frac{1}{2}dt) = q(t) + \frac{1}{2}\mathbf{Q}(t)\omega'(t)dt \tag{10.29}$$

Here J^B represents the angular momentum of a rigid body transformed to the body frame, and ω^B represents the transformed angular velocity; q is the four-vector (q_0,q_1,q_2,q_3). Finally, the torques are determined by

$$N_J = \sum_{i \in J}(r_i - R_J) \times f_i \tag{10.30}$$

Again, all frictional terms are embedded in the force f_J. These half-step calculations are necessary for the final, full-step update:

$$J(t+\frac{1}{2}dt) = J(t-\frac{1}{2}dt) + N(t)dt \tag{10.31}$$

$$J^B(t+\frac{1}{2}dt) = \mathbf{A}(t+\frac{1}{2}dt)J(t+\frac{1}{2}dt) \tag{10.32}$$

$$\omega_\alpha^B(t+\frac{1}{2}dt) = J_\alpha^B(t+\frac{1}{2}dt)/I_\alpha \qquad \alpha \in (x,y,z) \tag{10.33}$$

$$q(t+dt) = q(t) + \mathbf{Q}(t+\frac{1}{2}dt)\omega'(t+\frac{1}{2}dt)dt \tag{10.34}$$

The velocities of individual atoms, which are necessary for the friction evaluation, are determined using the angular velocities of the rigid bodies. The velocity due to rotation alone is determined in the body frame, then transformed into the lab frame and added to the translational velocity of the center of mass:

$$v_i(t) = V_J(t) + \mathbf{A_J}^{-1}\begin{bmatrix} z_i^B\omega_y^B - y_i^B\omega_z^B \\ x_i^B\omega_z^B - z_i^B\omega_x^B \\ y_i^B\omega_x^B - x_i^B\omega_y^B \end{bmatrix} \tag{10.35}$$

Finally, the atom positions are updated for the next energy evaluation using the new center-of-mass positions and the new Euler-Cayley parameters:

$$r_i(t+dt) = R_J(t+dt) + \mathbf{A_J^{-1}} r_i^B \tag{10.36}$$

The rigid-body dynamics routine can be run in one of three modes: FREE, LANGevin, or TCOUpling. The FREE mode simply allows frictionless dynamics, the LANGevin mode adds friction- and bath-dependent random forces, and TCOUpling uses the method of Berendsen et al. (1984). The coordinates (and velocities) are affected by using a nonzero friction coefficient in the Langevin dynamics algorithm with zero random forces. The friction coefficient is computed by the program from the equation

$$b_i = b_i^I(T_o/T - 1). \tag{10.37}$$

where b_i^I is given by the FBETa atom property. Note that FBETa is by default zero; that is, FBETa has to be specified in order to use TCOUpling. The target temperature T_o is specified by the TBATh parameter.

If the simulation involves light rigid groups such as water molecules, it is recommended that a time step on the order of .25 fsec be used. For heavier groups, time steps as large as 20 or 30 fsec are adequate.

10.2.3 Syntax of the Dynamics Rigid Statement

This statement sets up various parameters for rigid-body dynamics and then executes the specified number of dynamics steps. Parts of the molecule that are not specified in any GROUp statement will remain fixed. The constraints fix statement (Section 8.1) has no influence on rigid-body molecular dynamics (except for the default group selection).

DYNAmics RIGId { <**dynamics-rigid-statement**> } **END** is invoked from the main level of X-PLOR.

<**dynamics-rigid-statement**> **::==**

ASCIi=<logical> writes a formatted (ASCII) coordinate trajectory file if ASCIi is TRUE; if ASCIi is FALSE, an unformatted (binary) file is written. In the former case, a scale factor s and an offset o are applied to the coordinates r (i.e., $r' = s * (r + o)$); the result is converted to an integer number and written to the ASCII file using the specified format. If a hexadecimal format is specified, care should be taken to ensure that the application of the scale factor s and the offset o results in positive coordinates during the whole course of the molecular dynamics simulation. Otherwise, FORTRAN format runtime errors will occur.

CSEL=<selection> selects atoms that are written to the coordinate trajectory file (default: all atoms except those fixed by the constraints fix statement).

DT=<real> assigns the time step Δt value for the finite-difference integration in psec (default: 0.001 psec).

DYNMODE=<string> selects from three possible rigid-body dynamics modes: FREE, LANGevin, or TCOUpling (default: FREE).

FORMat=<string> provides a FORTRAN format for the ASCIi=TRUE option. Only integer or hexadecimal formats are allowed. (default: 12Z.6, hexadecimal).

GROUp=<selection> selects atoms that will form a rigid group. Multiple specification of GROUps will define multiple rigid groups. The group selections have to be disjoint (default: one group consisting of all atoms except those that have been fixed by the constraints fix statement; the default is overwritten as soon as a GROUp statement is issued).

NPRInt=<integer> determines the frequency with which the energy is printed in standard output (default: 1, i.e., at every dynamics step).

NSAVC=<integer> determines the frequency with which the coordinates are written to the coordinate trajectory file (default: 0).

NSAVV=<integer> determines the frequency with which the velocities are written to the velocity trajectory file (default: 0).

NSTEp=<integer> determines the number of steps (default: 100).

NTRFrq=<integer> determines if the overall center-of-mass translation and rotation of all "free" atoms is initially removed (default: 99999, i.e., remove the overall center-of-mass motion at the start).

OFFSet=<real> provides an offset o for the ASCIi=TRUE option (default: 800).

SCALe=<real> provides a scale factor s for the ASCIi=TRUE option (default: 10000).

TBATh=<real> specifies the temperature of the heatbath or the target temperature of the temperature-coupling method.

TRAJectory=<file> names coordinate trajectory file (default: none).

VASCii=<logical> writes a formatted (ASCII) velocity trajectory file if VASCii is TRUE; if VASCii is FALSE, an unformatted (binary) velocity trajectory file is written. In the former case, a scale factor s and an offset o are applied to the velocities r (i.e., $r' = s_v * (r + o_v)$); the result is converted to an integer number and written to the ASCII file using the specified format. If a hexadecimal format is specified, care should be taken to ensure that the application of the scale factor s_v and the offset o_v results in positive velocities during the whole course of the molecular dynamics simulation. Otherwise, FORTRAN format runtime errors will occur.

VELOcity=<file> names the velocity trajectory file (default: none).

VFORmat=<string> is the FORTRAN format of velocity trajectory file for the VASCii= TRUE option. Only integer or hexadecimal formats are allowed. (default: 12Z.6, hexadecimal).

VOFFset=<real> provides an offset o_v for the VASCii= TRUE option (default: 300).

VSCAle=<real> provides a scale factor s_v for the VASCii=TRUE option (default: 10000).

VSEL=<selection> selects atoms that are written to the velocity trajectory file (default: all atoms except those fixed by the constraints fix statement).

10.2.4 Requirements

The molecular structure, parameters, and atomic coordinates have to be defined. The constraints fix statement does not influence the propagation of coordinates. It will affect only the default selection of a rigid group and the periodic center-of-mass motion removal option.

10.2.5 Example: Run a Rigid-Body Dynamics Simulation

```
vector do (vx=maxwell(300.))  ( all )
vector do (vy=maxwell(300.))  ( all )
vector do (vz=maxwell(300.))  ( all )

dynamics rigid
   nstep=1000
   dt=0.001
   group=( residue 1:6 )   group=( residue 7:19 )
   group=( residue 20:22 ) group=( residue 23:30 )
   dynmode=tcou
   tbath=298.
   nprint=25
end
```

11 Management of Trajectories

The history of molecular dynamics calculations can be stored in the form of coordinate or velocity trajectory files. This chapter deals with reading, writing, manipulating, and analyzing molecular dynamics trajectories.

11.1 Trajectory Definitions

11.1.1 ASCII Trajectory Files

The first line of a trajectory file defines the number of comment records. This is followed by "header" records.

```
CORD     1000   1000  0      5932  0.204548E-01
^ type   ^begin ^skip ^void  ^number of atoms
                                  ^ timestep

  10000.0       800.000    12Z6
  ^ scale       ^ offset   ^ format

174      1     21     23     27     28     39
^number of free atoms
         ^      ^      ^      ^     ^ ... Ids of free atoms
```

Header records are in turn followed by coordinate sets separated by the string CORD. The first coordinate set contains all atoms. The subsequent coordinate sets contain only the selected atoms (see, for example, the CSELection statement in Section 10.1.6).

11.1.2 Binary Trajectory Files

The file format is identical to the CHARMm-DCD format (Brooks et al. 1983) , which can be read by QUANTA and a variety of other programs. The only exception is that the number of coordinate sets written to the

trajectory file is explicitly written into the header of CHARMm-DCD files, whereas XPLOR writes a zero instead.

11.1.3 Trajectory Statement

The following syntactic definition is used in many molecular dynamics application statements:

<trajectory-statement>:==

ASCIi=<logical> specifies whether input trajectory file(s) are ASCII (ASCIi=TRUE) or binary (ASCIi=FALSE). This statement must be issued before the INPUT files are specified to enable proper opening of the input files. (Files are opened in the specified mode as soon as an INPUt statement is issued.)

BEGIn=<integer> specifies the first frame to be read, in integration step units. This number can refer to any frame of the trajectory; i.e., reading can begin at any place in the trajectory file.

INPUt=<filename> specifies the name(s) of the trajectory file(s); several input files can be specified and read in the given order.

SKIP=<integer> specifies the number of frames to be skipped between successive readings, in integration step units. This must be a multiple of the frequency with which the trajectory was saved (e.g., NSAVC in Section 10.1.6).

STOP=<integer> specifies the last frame to be read from the trajectory file, in integration step units. It has to be equal to or smaller than the total number of integration steps saved in the trajectory file.

BEGIn, SKIP, and STOP refer to integration step units.

11.1.4 Diagnostic Error Messages

The following list contains the most common error messages encountered during reading of trajectory files and the possible causes of them:

READC-ERROR file 1 at step 0 means that one should check whether ASCIi is set to the right value.

READC: number of atoms do not match means that the molecular structure information and the trajectory file are not compatible.

READC: header mismatch on file means that the header of a trajectory file does not match the header of the previously read trajectory file.

READC: SKIP not compatible. Modified means that the SKIP value in a trajectory statement is not an integer multiple of saving frequency (e.g., NSAVc in Section 10.1.6) of the trajectory file or that SKIP is smaller than 0. The value of SKIP is adjusted to make it compatible with the saving frequency in the trajectory file.

READC: STOP not compatible. Modified means that the difference (STOP−BEGIN) in a trajectory statement is not an integer multiple of SKIP or that STOP is smaller than BEGIN. X-PLOR will attempt to adjust the value of STOP.

READC: EOF on file=',UNIT,' while trying to read step=',ISTEP means that the end of the trajectory file has been reached. This happens naturally at the end of each input file if several input files are used.

READC-ERROR file 1 at step ... means that something terribly wrong has occurred.

11.2 Reading Trajectories

The read trajectory statement allows one to extract single coordinate sets from a trajectory. It also makes it possible to loop through all or selected frames of a trajectory file and to carry out operations on each frame using X-PLOR's shell language. To facilitate shell-language control of the reading of trajectory files, a symbol $STATUS (of type string) is declared by the read trajectory statement. After an initial call to the read trajectory statement to define the input files and their format, the symbol $STATUS is set to "READ". The subsequent coordinate sets are read via READ DYNAmics next statements. After the last trajectory frame has been read, the symbol $STATUS is set to "COMPLETE".

11.2.1 Syntax

READ TRAJectory { <**read-trajectory-statement**> } **END**

<**read-trajectory-statement**> :==

<**trajectory-statement**> should be specified only for the initial call, to set the trajectory filenames and formats (see Section 11.1.3).

NEXT should be specified for subsequent calls.

11.2.2 Example

The example below reads frames from a fictitious molecular dynamics trajectory of two files until the last frame is reached:

```
1 read trajectory
```

```
2
3   asci=true
4   input=pti_00_50.crd
5   input=pti_50_100.crd
6   begin=1000 skip=1000 stop=100000
7
8 end
9
10 while ($status # "COMPLETE") loop traj
11
12   read trajectory next end
13
14 end loop traj
```

11.3 Writing Trajectories

The write trajectory statement allows one to construct "fake" trajectory files from single coordinate sets using the X-PLOR shell language.

11.3.1 Syntax

WRITe TRAJectory { <**write-trajectory-statement**> } **END** is invoked from the main level of XPLOR

<**write-trajectory-statement**> :==

ASCIi=<logical> specifies whether the output trajectory file is ASCII (ASCIi=TRUE) or binary (ASCIi=FALSE). A scale factor s and an offset o are applied to the coordinates before they are written to an ASCII file; i.e., $r' = s * r + o$. The result is converted to an integer number. (default: FALSE).

FORMat=<string> is the FORTRAN format for the OASCii= TRUE option. If the hexadecimal format is specified when writing an ASCII trajectory file, care should be taken to ensure that the coordinates after application of the scale factor s and the offset o are all positive. Otherwise, FORTRAN format runtime errors will occur during writing or reading of the ASCII file. The default offset and scale should work well in most cases. Only integer or hexadecimal formats are allowed. (default: 12Z.6, hexadecimal).

NEXT is to be used only for subsequent calls. It appends the current coordinate set to the trajectory file.

OFFSet=<real> provides an offset o for the OASCii=TRUE option (default: 800).

OUTPut=<filename> designates an output trajectory filename.

RESEt terminates writing to the trajectory file and begins writing to another trajectory file.

SCALe=<real> provides a scale factor s for the OASCii=TRUE option (default: 10000).

SELEction=<selection> specifies that only selected atoms are being written.

The integration step unit of the "fake" trajectory file that is produced by this statement is set to 1.

11.3.2 Requirements

The molecular structure information has to be present. The number of atoms in the molecular structure database has to match the number of atoms in the trajectory file.

11.3.3 Example

The following example reads a set of PDB coordinate files and merges them into a single trajectory file. Note that only X,Y,Z coordinates are saved in the trajectory file. The B and Q column information in the PDB file is lost.

```
evaluate ($count=0)
for $1 in ( a.pdb b.pdb c.pdb d.pdb e.pdb ) loop main
    evaluate ($count=$count+1)
    if ($count=1) then
        write trajectory
             output=trajectory.dcd
             ascii=false
        end
    else
        write trajectory
            next
        end
    end
    write trajectory
         reset
    end
end loop main
```

11.4 Merging Trajectories

The dynamics merge statement allows one to read a series of trajectory files specified in the trajectory statement, merges the frames, and writes them to the specified output file. Optionally, a least-squares fit of each frame is applied to the main coordinate set.

The atoms that are written to the output trajectory file can be limited to a subset of the atoms in the input files. The atoms to be left out in writing the new trajectory file have to be fixed by selecting them in a CONStraints FIX statement before the dynamics merge statement is issued.

The dynamics merge statement can also be used to convert the format of trajectory files from ASCII to binary and back.

11.4.1 Syntax

DYNAmics MERGe { <**dynamics-merge-statement**> } **END** is invoked from the main level of X-PLOR

<**dynamics-merge-statement**> **:==**

<**trajectory-statement**> is defined in Section 11.1.3.

ENSEmble=<**logical**> for ENSEmble=TRUE bypasses the mismatch checking between subsequent trajectory files. In this way it is possible to generate a pseudotrajectory from a group of otherwise unrelated trajectory files for the same system. One example where this option is useful consists of the merging of several independent molecular dynamics simulations in order to compute time-independent properties.

FORMat=<**string**> is the FORTRAN format for the OASCii=TRUE option. If the hexadecimal format is specified when writing an ASCII trajectory file, care should be taken to ensure that the coordinates after application of the scale factor s and the offset o are all positive during the whole course of the molecular dynamics simulation. Otherwise, FORTRAN format runtime errors will occur during writing or reading of the ASCII file. This is of particular concern when one is writing velocities. The default format and scale work well for Cartesian coordinates but do not work for velocities. Only integer or hexadecimal formats are allowed. (default=12Z.6, hexadecimal).

OASCii=<**logical**> writes a formatted (ASCII) file if OASCii is TRUE; if OASCii is FALSE, an unformatted (binary) file is written. A scale factor s and an offset o are applied to the coordinates r before they are written to an ASCII file; i.e., $r' = s * r + o$. The result is converted to an integer number.

OFFSet=<**real**> provides an offset o for the OASCii= TRUE option (default: 800).

ORIEnt { <**orient-statement**> } **END** fits the input trajectory frames to the main coordinate set using a least-squares fitting procedure and then writes the frames to the trajectory output file.

OUTPut=<**filename**> designates an output file for the trajectory (default: OUTPUT).

SCALe=<**real**> provides a scale factor s for the OASCii=TRUE option (default: 10000).

<**orient-statement**>**:==**

MASSweighting=<**logical**> is a logical flag indicating whether mass-weighting is applied for the least-squares fitting procedure (default: FALSE).

NOROtation=<logical> is a logical flag indicating whether a rotation is applied to the coordinates. If FALSE, only a translation vector is applied (default: TRUE).

SELEction=<selection> specifies the atoms that are used to define the least-squares transformation. However, the transformation is applied to all coordinates (default: (ALL)).

WEIGhting-with-B=<logical> is a logical flag indicating whether the B-array is used for weighting the atoms for the least-squares fitting procedure (default: FALSE).

11.4.2 Requirements

Before the dynamics merge statement can be used, the molecular structure of the molecule has to be defined by a structure statement. The number of atoms in the molecular structure database has to match the number of atoms in the trajectory file. If SKIP, STOP, and BEGIn are not compatible, the program issues a warning message and tries to adjust these parameters automatically.

11.4.3 Examples

The trajectory output data were originally stored in two files. The first file contains the first 50 psec, the second file the second 50 psec. These two files are then combined into one file.

```
structure @pti.psf end

dynamics merge

  ascii=true
  input=pti_00_50.crd
  input=pti_50_100.crd
  begin=1000 skip=1000 stop=100000

  oasci=true
  output=pti_00_100.crd

end
```

The next example writes only the backbone coordinates to a new file at every 5000 steps:

```
structure @pti.psf end

constraints fix
  (not((segid 4pti) and (name ca or name cb or name n))
end

dynamics merge

  asci=true
  input=pti_00_100.crd
  begin=5000 skip=5000 stop=100000

  oasci=false
```

```
14   output=pti_00_100.bb.crd
15
16 end
```

11.5 Analysis of Trajectories

A number of different tools for the investigation of molecular dynamics trajectories are provided by the dynamics analysis statements. These tools include different spatial and time averages as well as several different correlation functions. For all options, the input data are defined by a trajectory statement (see Section 11.1.3).

DYNAmics ANALyze <dynamics-analysis-statement>END is invoked from the main level of X-PLOR.

<dynamics-analysis-statement>:==

<dynamics-adf-statement> see Section 11.11

<dynamics-average-statement> see Section 11.6

<dynamics-density-statement> see Section 11.7

<dynamics-covariance-statement> see Section 11.8

<dynamics-power-statement> see Section 11.12

<dynamics-pick-statement> see Section 11.13

<dynamics-rdf-statement> see Section 11.10

<dynamics-time-correlation-statement> see Section 11.9

11.6 Average Coordinates and Fluctuations

The dynamics average statement computes the average coordinates and fluctuations of the rms displacements of a dynamics trajectory with respect to the average coordinates or with respect to the comparison coordinate set. The average coordinate are stored in the atom properties X,Y,Z , and the rmsds are stored in B.

11.6.1 Syntax

DYNAmics ANALyze AVERage { **<dynamics-average-statement>** } **END** is invoked from the main level of XPLOR.

<dynamics-average-statement>:==

<trajectory-statement> defines input data (see Section 11.1.3).

ORIEnt { <**orient-statement**> } **END** fits the input trajectory frames to selected atoms in the main coordinate set using a least-squares fitting procedure.

REFErence=AVERage | **COMParison** determines the reference coordinate set that is used to compute the rmsds (default: average).

Remark: the results are stored in X, Y, Z, B

<**orient-statement**>:==

MASSweighting=<**logical**> is a logical flag indicating whether mass-weighting is applied for the least-squares fitting procedure (default: FALSE).

NOROtation=<**logical**> is a logical flag indicating whether a rotation is applied to the coordinates. If FALSE, only a translation vector is applied (default: TRUE).

SELEction=<**selection**> specifies the atoms that are used to define the least-squares transformation. However, the transformation is applied to all coordinates (default: (ALL)).

WEIGhting-with-B=<**logical**> is a logical flag indicating whether the B-array is used for weighting the atoms for the least-squares fitting procedure (default: FALSE).

11.6.2 Example

The following example evaluates the average coordinates and rmsds:

```
1  structure @pti.psf end
2
3  dynamics analysis average
4
5    asci=true
6    input=pti_00_100.crd
7    begin=2000 skip=2000 stop=100000
8
9    reference=average
10
11   orient selection=(name ca or name c or name n) end
12
13 end
```

11.7 Density Analysis

The dynamics density statement computes the density of selected atoms in a spherical region around a reference point in shells of equal volume during a molecular dynamics run.

11.7.1 Syntax

DYNAmics { **ANALyze DENSity** <**dynamics-density-statement**> } **END** is invoked from the main level of X-PLOR.

<**dynamics-density-statement**> **:==**

<**trajectory-statement**> defines input data; see Section 11.1.3.

MINImum-image BOXX=<**real**> **BOXY=**<**real**> **BOXZ=**<**real**> **END** defines x,y,z dimensions of a rectangular box for periodic boundary conditions (default: no periodic boundary conditions).

ORIGin=<**vector**> provides the origin of a sphere (default: (0 0 0)).

OUTPut=<**filename**> writes results to the specified file (default: OUTPUT).

RADIus=<**real**> provides the radius of a sphere (default: 0).

RGRId=<**real**> calculates an equal-volume profile in steps of RGRId (default: 2).

SPECies=<**selection**> selects atoms (default: (ALL)).

11.8 Covariance Analysis

The dynamics covariance statement computes the covariances of the spatial atom displacements of a dynamics trajectory for selected pairs of atoms. The function

$$s_{1,2} = \frac{< x[1] * x[2] >}{\sqrt{< x[1] * *2 >< x[2] * *2 >}} \tag{11.1}$$

is calculated for all atoms #1 in SET1 and #2 in SET2 where $x[\cdot]$ is the coordinate displacement with respect to the average, and the corresponding covariance matrix is written to a file.

11.8.1 Syntax

DYNAmics ANALyze COVAriance { <**dynamics-covariance-statement**> } **END** is invoked from the main level of X-PLOR.

<**dynamics-covariance-statement**> **:==**

<**trajectory-statement**> defines the input data; see Section 11.1.3.

SET1=<**selection**> selects "SET1"-species (default: (ALL)).

SET2=<**selection**> selects "SET2"-species (default: (ALL)).

OUTPut=<**filename**> writes the covariance matrix to the specified file (default: OUTPUT).

11.8.2 Example

The following example shows how to calculate the covariance matrix for all ca atoms:

```
structure @pti.psf end

dyamics analysis covariance

  asci=true
  input=pti_00_100.crd
  begin=1000 skip=1000 stop=100000

  set1=(name ca)
  set2=(name ca)

  output=pti_ca_covar.matrix

end
```

11.9 Time Correlation Analysis

The dynamics time correlation statement computes different time autocorrelation functions averaged over a set of atoms for NCORS time steps. It is possible to select the included atoms by definition of a shell around a reference point. To specify the time step for the correlation function averaging, one should use the SKIP parameter in the trajectory statement. If the number of selected atoms is exactly one, the long time limit of the correlation function is shifted to zero.

The following correlation functions can be evaluated: positional auto correlation function

$$\mathrm{AC}(\tau) =< R(t) * R(t+\tau) >, \tag{11.2}$$

mean square displacement correlation function

$$\mathrm{MSD}(\tau) =< (R(t) - R(t+\tau))^2 >, \tag{11.3}$$

and finite-difference velocity autocorrelation function

$$\mathrm{VAC}(\tau) =< V(t) * V(t+\tau) > \tag{11.4}$$

where $V(t) = \frac{dX(t)}{dt}$.

11.9.1 Syntax

The VAC autocorrelation function uses a finite difference approximation to compute the velocities from a coordinate trajectory file. If a velocity trajectory file is available, the AC auto-correlation function should be used.

DYNAmics ANALyze CORRelation { <**dynamics-time-correlation-statement**> } **END** is invoked from the main level of X-PLOR.

<dynamics-time-correlation-statement> :==

<trajectory-statement> defines the input data; see Section 11.1.3.

FUNCtion=<AC|MSD|VAC> specifies the type of correlation function to be evaluated.

LENGth=<real> gives the length of correlation function in psec.

NORMalize=<logical> normalizes a correlation such that c(0)=1.0.

OUTPut <filename> writes the correlation function to the specified file.

SPECies=<selection> gives an average (center of mass) over selected atoms.

SHELl { <shell-statement> } END provides an additional selection based on dynamical position.

<shell-statement>:==

<trajectory-statement>

CUTHigh=<real>

CUTLow=<real>

MODE=<SELF|SIMUltaneously>

ORIGin=<vector>

11.10 Radial Distribution Functions

The analysis radial distribution statement computes $G_{ab}(r)$ radial distribution functions of B-species atoms around A-species atoms and the coordination numbers of B-species around A-species. Periodic boundary conditions can be taken into account by activating the minimum image convention.

11.10.1 Syntax

DYNAmics ANALysis RDF { <dynamics-rdf-statement> } END is invoked from the main level of X-PLOR.

<dynamics-rdf-statement> :==

<trajectory-statement> defines the input data; see Section 11.1.3.

ASPEcies=<selection> selects "A"-species.

BSPEcies=<selection> selects "B"-species.

DENSity=<real> adjusts the density to normalize the distribution (default: 1).

MINImum-image BOXX=<real> BOXY=<real> BOXZ=<real> END defines x,y,z dimensions of a rectangular box for periodic boundary conditions (default: no periodic boundary conditions).

MODE=<AVERage|CENTer|POINt> provides alternate ways of computing radial distributions. AVERage computes radial distributions around each A-species atom within RADIus around ORIGin and averages them. CENTer computes the distribution around the geometric center of all A-species. POINt does it around the fixed ORIGin.

ORIG=<vector> is explained under MODE (default: (0 0 0)).

OUTPut=<filename> writes the radial distribution function to the specified file (default: OUTPUT).

RADIus=<real> is explained under MODE (default: 0).

RGRID=<real> is the interval between values of r for which G_{ab} is evaluated (default: 0.05).

RMAX=<real> provides the maximum value of r for which radial distribution function is calculated (default: 10.0).

11.11 Angular Distribution Functions

The dynamics angular distribution statement computes angular (dipole) distribution functions of B-species around A-species. B-species defines the dipoles on a group basis: for each selected group, a dipole is defined according to the atom property CHARge and the main coordinates. The dipole is positioned at the center of mass (SCALar MASS) of the selected group. The dipoles are calculated over the atoms of selected groups, and the dipole position is set equal to the center of mass of the group. RGRID is the radial grid spacing and AGRID the angular spacing. The calculation is performed in elements of equal volume.

11.11.1 Syntax

DYNAmics ANALysis ADF { <dynamics-adf-statement> } END is invoked from the main level of X-PLOR.

<dynamics-adf-statement> :==

<trajectory-statement> defines the input data; see Section 11.1.3.

AGRId=<real> specifies an angular grid (default: 10.0).

ASPEcies=<selection> selects "A"-species.

BSPEcies=<selection> selects "B"-species.

DENSity=<real> is the density that normalizes the distribution (default: 0.0334).

MINImum-image BOXX=<real> BOXY=<real> BOXZ=<real> END defines x,y,z dimensions of a rectangular box for periodic boundary conditions (default: no periodic boundary conditions).

MODE=<AVERage|CENTer|POINt> provides alternate ways of computing angular distributions. AVERage computes distributions around each A-species atom within RADIus around ORIGin and averages them. CENTer computes the distribution around the geometric center of all A-species. POINt does it around the fixed ORIGin

ORIGin=<vector> is explained under MODE (default: (0 0 0)).

OUTPut=<filename> writes the radial distribution function to the specified file (default: OUTPUT).

RADIus=<real> is explained under MODE (default: 0).

RGRID=<real> is the interval between values of r for which G_{ab} is evaluated (default: 0.5).

RMAX=<real> provides a radial distribution window upper limit for r (default: 10.0).

RMIN=<real> provides a radial distribution window lower limit for r (default: 2.0).

11.12 Power Spectrum Analysis

The analysis power statement computes a power spectrum of a vector time series by fast Fourier transformation (FFT). Optionally, the spectrum is trimmed or the autocorrelation function is computed by Fourier back-transformation.

11.12.1 Syntax

DYNAmics ANALysis POWEr { <dynamics-power-statement> } END is invoked from the main level of X-PLOR.

<dynamics-power-statement> :==

<trajectory-statement> defines the input data; see Section 11.1.3.

BACK=<logical> performs Fourier back-transformation on power spectrum to obtain the vector autocorrelation function (default: FALSE).

$$C(\tau) = < \vec{r}(t) * \vec{r}(t+\tau) > \tag{11.5}$$

CLENgth=<realps> provides length of correlation function in units of psec; it affects only the output file (default: 10).

COUTput=<filename> writes the correlation function to the specified file.

PLENgth=<real> provides the length of the power spectrum in units of psec^{-1}; it affects only the output file, not the calculation (default: 1000).

POUTput=<filename> puts the power spectrum into the specified file.

POWER-2=<integer> is an optional specification of a power of 2 for the FFT (default: 1).

PSKIp=<int> is the skip value for the printout of the power spectrum. It does not affect the calculation. If pskip is zero, no power spectrum is written (default: 1).

SPECies=<selection> gives an average (geometric center) over selected atoms (default: no atoms selected).

TRIM=<logical> trims the first (largest) peak from the power spectrum if TRIM is TRUE; if a correlation function is calculated later and the peak was at 0, this is equivalent to shifting the long time limit of the correlation function to 0 ($\delta(0) \stackrel{FFT}{\rightarrow}$ additive constant) (default: FALSE).

WINDow=(<real> <real>) sets to zero everything in the power spectrum that is outside the window defined by the two values in units of psec^{-1} (default: (0,1000)).

11.13 Picking Properties for Trajectories

The pick statement allows one to extract time series of terms from a trajectory.

11.13.1 Syntax

DYNAmics ANALysis PICK { **<dynamics-pick-statement>** } **END**
is invoked from the main level of X-PLOR.

<dynamics-pick-statement> :==

<trajectory-statement> defines the input data; see Section 11.1.3.

<pick-statement> is explained in Section 5.1.2.

OUTPut=<filename> is the name of the file that contains the time series.

12 Crystallographic Diffraction Data

The xrefin statement sets up information that is needed for all aspects of X-PLOR dealing with crystallographic diffraction data.

12.1 Crystallographic Target Functions

X-PLOR provides several possibilities for the effective energy E_{XREF}. The selection of the target is specified by the TARGet keyword. There are seven possible choices: RESIdual, AB, F1F1, F2F2, E1E1, E2E2, and PACKing.

(12.1)

$$E_{XREF} = \begin{cases} W_A/N_A \sum_{\vec{h}} w_{\vec{h}}[|F_{obs}(\vec{h})| - k|F_c(\vec{h})|]^2 \quad + E^P_{XREF} \\ W_A/N_A \sum_{\vec{h}} w_{\vec{h}}[A_{obs}(\vec{h}) - kA_c(\vec{h})]^2 + w_{\vec{h}}[B_{obs}(\vec{h}) - kB_c(\vec{h})]^2 \\ W_A(1 - \mathrm{Corr}[|F_{obs}(\vec{h})|\ ,\ |F_c(\vec{h})|]) \\ W_A(1 - \mathrm{Corr}[F_{obs}(\vec{h})^2\ ,\ F_c(\vec{h})^2]) \\ W_A(1 - \mathrm{Corr}[|E_{obs}(\vec{h})|\ ,\ |E_c(\vec{h})|]) \\ W_A(1 - \mathrm{Corr}[E_{obs}(\vec{h})^2\ ,\ E_c(\vec{h})^2]) \\ W_A(1 - \mathrm{Pack}) \end{cases}$$

$\vec{h} = (h, k, l)$ is the Miller indices of the selected reflections, F_{obs} is the observed structure factors, F_c is the computed structure factors, A_{obs} and A_c are the real components, B_{obs} and B_c are the imaginary components of the structure factors, N_A is a normalization factor, k is a scale factor, W_A is an overall weight, $w_{\vec{h}}$ is the individual weights of the reflections, Es are normalized structure factors, and "Corr" is the standard linear correlation coefficient. The computation of the effective energy E_{XREF} is accompanied by printing the unweighted R value

$$R = \frac{\sum_{\vec{h}} ||F_{obs}(\vec{h})| - k|F_c(\vec{h})||}{\sum_{\vec{h}} |F_{obs}(\vec{h})|} \tag{12.2}$$

for the first choice in Eq. 12.1, the unweighted vector R value

$$R^{vec} = \frac{\sum_{\vec{h}} \sqrt{(A_{obs}(\vec{h}) - kA_c(\vec{h}))^2 + (B_{obs}(\vec{h}) - kB_c(\vec{h}))^2}}{\sum_{\vec{h}} |F_{obs}(\vec{h})|} \tag{12.3}$$

for the second choice, or the various correlation coefficients for the third to sixth choices. The R values are stored in the symbol \$R, and the correlation coefficients are stored in the symbol \$CORR. If the data are partitioned into a test and a working set (see Chapter 15), the corresponding values for the test set are stored in the symbols \$TEST_R and \$TEST_CORR.

The selection of reflections is accomplished by the RESOlution and FWINdow statements (see below). "Corr" is defined through

$$\mathrm{Corr}[x\ ,\ y] = \frac{< xy - < x >< y >>}{\sqrt{< x^2 - < x >^2 >< y^2 - < y >^2 >}} \tag{12.4}$$

where the angle brackets denote a weighted ($w_{\vec{h}}$) averaging over all selected Miller indices $\vec{h}$. $F_c(\vec{h})$ is defined as

$$F_c(\vec{h}) = F_{calc}(\vec{h}) + F_{part}(\vec{h}) \tag{12.5}$$

where $F_{part}(\vec{h})$ is "partial" structure factors that can be used to represent a "frozen" part of the molecule or bulk solvent contributions, and $F_{calc}(\vec{h})$ represents the structure factors that are computed from the current atomic model. $w_{\vec{h}}$ provides individual weights for each reflection $\vec{h}$. The overall weight W_A relates E^A_{XREF} to the other energy terms (see Section 4.6).

The normalized structure factors (Es) are computed from the structure factors (Fs) by averaging the Fs in equal reciprocal volume shells within the specified resolution limits. The number of shells is specified by MBINs.

The purpose of the normalization factor N_A (first and second choice in Eq. 12.1) is to make the weight W_A approximately independent of the resolution range during SA-refinement. N_A has been set to $N_A = \sum_{\vec{h}} W_{\vec{h}} |F_{obs}(\vec{h})|^2$. The scale factor k in Eq. 12.1 is set to

$$k = \sum_{\vec{h}} W_{\vec{h}} |F_{obs}(\vec{h})||F_c(\vec{h})| / (\sum_{\vec{h}} W_{\vec{h}} F_c(\vec{h})^2) \tag{12.6}$$

unless it is set manually by the FFK statement. Eq. 12.6 is a necessary condition to minimize the residual.

The term E^P_{XREF} represents phase restraints if W_P is set to a nonzero number.

$$E^P_{XREF} = W_P / N_P \sum_{\vec{h}} W_{\vec{h}} S\{modulo_{2\pi}(\phi_{obs}(\vec{h}) - \phi_c(\vec{h})), a\cos(m(\vec{h}))\} \tag{12.7}$$

N_P is a normalization factor set equal to the number of phase specifications occurring in the sum, $\phi_{obs}(\vec{h})$ is the phase centroid obtained from mir or other methods (PHASe specifications; see Section 12.4), $\phi_c(\vec{h})$ is the phase of the calculated structure factors $F_c(\vec{h})$, $m(\vec{h})$ is the individual figure of merit (FOM specifications; see Section 12.4), and $S\{x, y\}$ is a well function with harmonic "wells" given by

$$S\{x,y\} = \begin{cases} (x-y)^2 & \text{x > y} \\ 0 & \text{-y < x < y} \\ (y+x)^2 & \text{-y > x} \end{cases} \tag{12.8}$$

This form of the effective energy E^P_{XREF} ensures that the calculated phases are restrained to $\phi_{obs}(\vec{h})\pm\text{acos}(m(\vec{h}))$.

The structure factors ($F_{calc}(\vec{h})$) of the atomic model are given by

(12.9)

$$\begin{aligned} F_{calc}(\vec{h}) &= \sum_{s\in S}\sum_{n\in NCS}\sum_{i} Q_i f_i(\vec{h}) exp(-B_i(\mathcal{F}^*\vec{h})^2/4) \\ &\quad \times exp(2\pi i\vec{h}\cdot(\mathcal{O}_s\mathcal{F}(\mathcal{O}_n\vec{r_i}+\vec{t_n})+\vec{t_s})) \end{aligned}$$

The first sum extends over all symmetry operators $(\mathcal{O}_s,\vec{t_s};s\in S)$ composed of the matrix $\mathcal{O}_s$ representing a rotation and a vector $\vec{t_s}$ representing a translation. The second sum extends over all non-crystallographic symmetry operators $(\mathcal{O}_n,\vec{t_n};n\in NCS)$ if they are present; otherwise only the identity transformation is used (see Chapter 16). The third sum extends over all unique atoms i of the system. The quantity $\vec{r_i}$ denotes the orthogonal coordinates of atom i in Å. $\mathcal{F}$ is the 3×3 matrix that converts orthogonal coordinates into fractional coordinates; $\mathcal{F}^*$ denotes the transpose of it. The columns of $\mathcal{F}^*$ are equal to the reciprocal unit cell vectors $\vec{a^*},\vec{b^*},\vec{c^*}$. Q_i is the occupancy for each atom. B_i is the individual atomic temperature factor for atom i. Both quantities correspond to the Q and B atom properties (Section 2.16), which can be read along with the atomic coordinates (see Section 6.1). The atomic scattering factors $f_i(\vec{h})$ are approximated by an expression consisting of four Gaussians and a constant

$$f_i(\vec{h}) = \sum_{k=1}^{4} a_{ki} exp(-b_{ki}(\mathcal{F}^*\vec{h})^2/4) + c_i + \text{i}d_i \qquad (12.10)$$

The constants a_{ki} and b_{ki} are specified in the **SCATter** statement. A file containing the most common atomic scattering factors is stored in \$SYMMETRY/symmetry.list; the listing is obtained from the *International Tables for Crystallography* (Hahn ed. 1987). The term id_i denotes an imaginary constant that can be used to model anomalous scattering. Usually, d_i is referred to as f''. There is no explicit f' constant in X-PLOR. Rather the f' constant needs to be added to the c_i constant.

Eq. 12.9 represents the space-group general form of the "direct summation" formula, which is used to compute the structure factors. The fast Fourier transformation (FFT) method consists of computing $F_{calc}(\vec{h})$ by numerical evaluation of the atomic electron density on a finite grid followed by an FFT. The FFT method provides a way to speed up the calculation. The METHod statement can be used to switch between the FFT method and the direct summation method.

An approximation is used to reduce the computational requirements when multiple evaluations of Eq. 12.1 are required. The approximation involves not computing $F_{calc}(\vec{h})$ and its first derivatives at every dynamics

or minimization step. The first derivatives are kept constant until any atom has moved by more than Δ_F (TOLErance in xrefin statement) relative to the position at which the derivatives were last computed. At that point, all derivatives are updated. Typically, Δ_F is set to 0.2 Å for dynamics and to 0–0.05 Å for minimization.

The PACKing target is defined for evaluating the likelihood of packing arrangements of the search model and its symmetry mates in the crystal (Hendrickson and Ward 1976). A finite grid that covers the unit cell of the crystal is generated. The grid size is specified through the GRID parameter in the xrefin FFT statement. All grid points are marked that are within the van der Waals radii around any atom of the search model and its symmetry mates. The number of marked grid points represents the union of the molecular spaces of the search model and its symmetry mates. Maximization of the union of molecular spaces is equivalent to minimization of the overlap. Thus, an optimally packed structure has a maximum of the packing function. "Pack" in Eq. 12.1 contains the ratio of the number of marked grid points to the total number of grid points in the unit cell. For instance, a value of 0.6 means 40% solvent contents. $E_{XREF} = W_A(1 - \text{Pack})$ is then set to 0.4 if $W_A = 1$.

For further reading on the crystallographic target functions in X-PLOR, see Brünger (1988, 1989, 1990).

12.2 Orthogonalization Convention

The X-PLOR orthogonalization/fractionalization convention defines a in same direction as x; b is chosen such that y is in the plane defined by a and b. This convention is used by all X-PLOR routines. To avoid difficulties when using a different convention, read the coordinates, fractionalize them according to your convention, and then orthogonalize them using the X-PLOR convention.

```
xrefin
{===>} coor @your_convention.coordinates
coor rotate
{===>}  matrix=( 0.0333 0.   0.0192     )            {*Use your convention.*}
               ( 0.     0.02 0.         )
               ( 0.     0.   0.0192     )
end
coor translate
    vector=( 0. 0. 0. )                           {*Translate if necessary.*}
end

{===>} xrefin a=30. b=50. c=60. beta=120  end                  {*Unit cell.*}

coor orthogonalize end
{===>} write coordinates output=xplor_convention.coordinates end
```

12.3 Syntax of the Xrefin Statement

The xrefin statement is used to read structure factors, symmetry operators, atomic-form factors, unit cell parameters, etc. It also branches into several other statements that allow one to manipulate structure factors, refine certain parameters, carry out translation or rotation searches, compute solvent masks, or compute electron density maps.

Manipulations of structure factors are carried out for the selected reflections. The selection of reflections is accomplished by the RESOlution and FWINdow statements. Atoms contributing to the structure factor F_{calc} are selected through the SELEction and SCATter statements.

XREFin { <**xrefin-statement**> } **END** is invoked from the main level of XPLOR.

<**xrefin-statement**>:==

A=<**real**> specifies a of unit cell (default: 1.0 Å).

ALPHa=<**real**> specifies α of unit cell (default: 90°).

B=<**real**> specifies b of unit cell (default: 1.0 Å).

BETA=<**real**> specifies β of unit cell (default: 90°).

C=<**real**> specifies c of unit cell (default: 1.0 Å).

DO <**xrefin-do-statement**> manipulates structure factors (see Section 12.5).

EXPAnd expands selected reflections to P1; i.e., the crystallographic symmetry operators and the Hermitian symmetry operator (if HERMitian=TRUE) are applied to the selected reflections. Multiple entries that emerge during this process are discarded. Multiple application of EXPAnd is possible. The second time around, it does not have any effect unless new data are added or more symmetry operators are added.

FFK=<**real**> sets k in Eq. 12.1 to the specified value. If FFK is set to 0, k is automatically determined by Eq. 12.6. Automatic scaling also declares symbols ($FFK, $TEST_FFK) that contain the values of k for the working set and the test set, respectively (default: 0, i.e., automatic scaling).

FFT { <**FFT-statement**> } **END** specifies parameters for the FFT method.

FWINdow <**real**> <**real**> sets F_{obs} amplitude limits for the selection of reflections. One of the <real> values is the upper value; the other one is the lower value; it does not matter whether the upper or the lower limit comes first (default: FWINdow 0 100000).

GAMMa=<**real**> specifies γ of unit cell (default: 90°).

GENErate complements the current reflections to yield a full asymmetric unit of reflections for the specified resolution range. If no current reflections are present, a full asymmetric unit is generated. The new Fobs are set to 1 except for systematic absences, in which case Fobs is set to 0. Fcalc, Fpart are set to 0; weight, sigma, and FOM are set to 1. Action is taken as soon as this statement is issued.

HERMitian=<logical> specifies whether Friedel mates are identical (HERMitian=TRUE) or different from each other (HERMitian=FALSE). Thus, for anomalous scattering data, one has to set HERMitian=FALSE, specify the imaginary scattering components, and specify the ASELection statement for the appropriate atoms (default: TRUE).

LOOKup=<logical> is a flag indicating whether to use lookup tables for the computation of the atomic form factors. The flag only applies to the FFT method It is recommended to turn off the lookup tables if problems with minimization procedures are encountered (default: TRUE).

MAP { <**xrefin-map-statement**> } **END** computes electron density maps (see Section 14.1).

MBINs=<integer> specifies the number of bins for the R value analysis (PRINt R), phase difference analysis (PRINt PHASe), data completeness analysis (PRINt COMPleteness), and computation of normalized structure factors (Es) (default: 8).

METHod= DIREct | FFT specifies choice of method to compute F_{calc} (default: FFT).

NREFlections=<integer> is a required parameter that allocates space for the reflections. It has to be greater than or equal to the actual number of reflections (default: 200).

OPTImize BFACtor { <**xrefin-optimize-bfactor-statement**> } **END** optimizes individual (restrained) isotropic B-factors (see Section 13.4).

OPTImize GROUp { <**xrefin-optimize-group-statement**> } **END** optimizes group B-factors and/or occupancies (see Section 13.3).

OPTImize OVERall { <**xrefin-optimize-overall-statement**> } **END** optimizes an overall isotropic or anisotropic B-factor (see Section 13.2).

PRINt COMPleteness prints the ratio of the number of observed reflections to the number of theoretically observable reflections ("completeness," printed as a percentage). The analysis is carried out as a function of resolution. The overall completeness is stored in the symbol $COMPLETENESS. If the data are partitioned into a test set and a working set, a completeness analysis is also carried out for the test reflections (see Chapter 15). In this case, the completeness for the test reflections (TEST=1) is stored in the symbol $TEST_COMPLETENESS, and the

completeness for the working set is stored in $COMPLETENESS. This statement also produces a listing that can be plotted by a Mathematica script.

PRINt PHASe prints the average phase difference as a function of resolution for the selected reflections and stores the overall phase difference in the symbol $DPHI. Phases differences are weighted with the WEIGht array (see Section 12.5). Only reflections with non-zero figure of merit (FOM) are considered in the phase difference average. If a figure-of-merit weighted average is required, the user should issue a "DO (WEIGHT=MAX(0,FOM))" statement; this will overwrite the existing weights. If the data are partitioned into a test set and a working set, a phase difference analysis for the test reflections is also carried out (see Chapter 15). The "free" phase difference is stored in the symbol $TEST_DPHI. Note that no prior update of F_{calc} is performed, and thus the user must issue an UPDAte statement if the F_{calc} array is undefined or not well defined. This statement also produces a listing that can be plotted by a Mathematica script.

PRINt R prints the R value as a function of resolution for the selected reflections and stores the overall R value in the symbol $R. If the data are partitioned into a test set and a working set, a free R value analysis is also carried out (see Chapter 15). The free R value is stored in the symbol $TEST_R. Note that no prior update of F_{calc} is performed. Thus, the user must update F_{calc} with the UPDAte statement if the F_{calc} array is undefined or if an energy calculation (e.g., energy minimization or molecular dynamics) was carried out previously. This statement also produces a listing that can be plotted by a Mathematica script.

PRINt TARGet prints the value of E_{XREF} (Eq. 12.1). If the residual or the AB vector-residual is chosen, the R value is stored in the symbol $R. If targets are chosen that contain a correlation coefficient, its value is stored in the symbol $CORR. If the data are partitioned into a test and a working set (see Chapter 15), the corresponding values for the test set are stored in the symbols $TEST_R and $TEST_CORR. Note that no prior update of F_{calc} is performed, and thus the user must issue an UPDAte statement if the F_{calc} array is undefined or not well defined.

PRINt WILSon makes a Wilson plot (Wilson 1949; Rogers 1965; Main 1975). The overall scale factor and B-factor are obtained by a least-squares fit of $\log(I_{obs}/f^2)$ *vs.* s^2. The result is stored in symbols $BFACTOR, $SCALE, and $TEST_BFACTOR, $TEST_SCALE for the working and test sets, respectively. This statement also produces a listing that can be plotted by a Mathematica script.

REDUce decreases selected reflections to an asymmetric unit. In the case of multiple entries, the first entry is kept and the duplicates are discarded; i.e., no data averaging is performed.

REFLection { <**xrefin-reflection-statement**> } **END** initiates reading or merging of diffraction data (see Section 12.4).

RESEt erases the current xrefin database, i.e., the atomic-form factors, symmetry operators, reflections, and unit cell parameters.

RESOLution <**real**> <**real**> sets resolution limits in Å for the selection of reflections. One of the <real> values is the high resolution limit, and the other one is the low resolution limit; it does not matter whether the high or the low resolution limit comes first. One of the limits can be set to the string "INFInity", e.g., "RESOlution INFInity 3". This statement will then include the 0,0,0 reflection in all calculations (default: RESOlution 10. 3.).

SCATter <**selection**> <**real**> <**real**> <**real**> <**real**> <**real**> <**real**> <**real**> <**real**> <**real**> [**IMAGinary** <**real**>] adds an atomic-form factor specification to the xrefin database. The statement specifies the coefficients $a_1, b_1, a_2, b_2, a_3, b_3, a_4, b_4, c, d$ (Eq. 12.10) for the selected atoms (default: none). An atom will contribute to the structure factors only if it has been selected in the SELEction statement. All selected atoms are required to be selected by one of the SCATter statements except for atoms with chemical type H*. Care should be taken not to produce an overlapping definition of atom selections; e.g., if there is a ca ion, one should exclude it from the atomic-form factor definition for carbon atoms. The optional IMAGinary parameter should be used for anomalous scatterers. Usually, this parameter is referred to as f''. There is no explicit f' constant in X-PLOR. Rather the f' constant needs to be added to the c constant.

"SCATter RESEt" will erase the existing atomic form factor entries.

SEARch ROTAtion { <**xrefin-search-rotation-statement**> } **END** is a translation search (see Section 17.3).

SEARch TRANslation { <**xrefin-search-translation-statement**> } **END** is a rotation search (see Section 17.3).

SELEction= <**selection**> selects atoms that will be used in the next structure factor calculation (default: (chemical H*)). An atom will contribute to the structure factor only if it has been selected in the SELEction statement. For all selected atoms, entries in the SCATter statements must exist except for atoms with chemical type H*. The selection remains active until a new SELEction statement is issued.

SOLMask { <**xrefin-solmask-statement**> } **END** computes a solvent mask (see Section 12.7).

SYMMetry=<**symmetry-operator**> adds a new symmetry operator $(\mathcal{O}_s, \vec{t}_s)$ to the xrefin database. The notation is the same as in the *International Tables for Crystallography* (Hahn ed. 1987), e.g.,

$$(-x, y + 1/2, -z). \tag{12.11}$$

Multiple entries specify the space group. A listing of the symmetry operators of all crystallographic space groups is stored in file "symlib.sym" in the "symmetry" directory. In the case of centered space groups, all symmetry operators have to be specified; otherwise symmetry images will be missing for the packing interactions. Normally, this redundant specification of symmetry operators represents only a small increase in CPU time. The option "SYMMetry RESEt" will erase the existing symmetry operators (default: symmetry=(x,y,z)).

TARGet= RESIdual | AB | F1F1 | F2F2 | E1E1 | E2E2 | PACKing specifies choice of the target function; see Eq. 12.1 (default: RESIdual).

TOLErance=<real> specifies the maximum amount Δ_F (in Å) by which any atomic coordinate can deviate from the position when F_{calc} was last computed during molecular dynamics, energy minimization, or energy calculations. If **TOLErance** is exceeded, F_{calc} and its derivatives with respect to atomic parameters are recomputed (default: 0.5 Å).

UPDAte computes F_{calc} for selected reflections using the current atomic model and atomic-form factors.

WA=<real> specifies the overall weight factor W_A in Eq. 12.1 (default: 1).

WP=<real> specifies the overall weight factor W_P for the phase term E^P_{XREF} in Eq. 12.1 (default: 0).

WRITe REFLection { **<write-reflection-statement>** } **END** writes the specified xrefin properties, such as FOBS or FCALC, of all selected reflections to a specified output file (see Section 12.4).

<FFT-statement>:==

AVOID=<integer> facilitates avoidance of the specified integer as a factor in the x and y physical dimension of the 3-d electron density matrix in order to avoid memory conflicts on certain supercomputers (default: machine dependent, set automatically).

BASE=<integer> specifies the minimum prime allowed in FFT dimensions (default: machine dependent, set automatically).

BSCALefactor=<real> sets the artificial temperature factor to minimize aliasing effects (default: 20.0 Å^2).

ELIMit=<real> defines an atomic "radius" for the electron density calculation by specifying the ratio of the atomic-form factor at zero and at the radius of the atom in natural logarithmic units (default: 7.00). The electron density outside the radius is set to zero.

GRIDsize=<real> specifies the grid size relative to the high resolution limit (default: 0.33, which corresponds to 1/3 of the high resolution limit).

MEMOry=<integer> indirectly determines the factorization of the FFT by specifying the maximum memory allocation allowed for the electron density matrix in units of double complex words (default: 500000).

PRIMe=<integer> specifies the maximum prime allowed for FFT dimensions (default: machine dependent, set automatically).

12.3.1 Requirements

The molecular structure and atomic coordinates have to be defined. The atom selections are fragile (Section 2.15).

12.4 Reflection Files

12.4.1 Syntax

WRITe REFLection { <**write-reflection-statement**> } **END** is invoked as an xrefin statement.

<**write-reflection-statement**>:==

OUTPut=<filename> is an output file (default: OUTPUT).

<**xrefin-property**> specifies xrefin properties (see Section 12.5) to be written to output file.

REFLection { <**xrefin-reflection-statement**> } **END** is invoked as an xrefin statement. It is important to specify the maximum expected allocation for reflections (NREFlection statement) before starting to read the reflections. Expansion of the reflections to P_1 is not required. Therefore, only the set of unique reflections should be read. To merge two data sets or to merge different information (such as F_{obs} and F_{calc}), the two reflection files should be read with separate reflection statements. Overlapping information that is specified in the most recent reflection statement will replace information that was already present in the specified Miller indices.

<**xrefin-reflection-statement**>:==

INDEx <**integer**> <**integer**> <**integer**> { <**reflection-property**> } specifies the Miller (h,k,l) indices for the current reflection. The reflection-properties comprise all information provided for this reflection. The input line echo is automatically turned off, except for the first INDEx statement.

RESEt erases the current reflection database.

<**reflection-property**>:==

FCALc <real> <real> specifies F_{calc} amplitude and phase (default: 0).

FOBS=<real> specifies F_{obs} amplitude.

FOM=<real> specifies figure of merit $m(\vec{h})$ (default: -1.0).

FPARtial=<real>=<real> specifies "partial" structure factor. Note that the partial structure factor is zero by default. It is always added to F_{calc} prior to evaluation of crystallograhic target functions, residuals, or correlation coefficients. It is not, however, automatically added to F_{calc} when performing a XREFin DO operation (Section 12.5) (default: 0).

PHASe=<real> specifies "observed" phase $\phi_{obs}(\vec{h})$.

SIGMa=<real> specifies the σ value (default: 1).

TEST=<integer> is used to partition the data into a working set and a test set for cross-validation (see Chapter 15) (default: 0).

WEIGht=<real> specifies the weight $W_{\vec{h}}$ (default: 1).

12.4.2 Example: A Crystallographic Reflection File

The input is free-field, and each reflection record may be extended over several lines. Only the information that is available or wanted should be specified.

```
INDEx 1 2 3  FOBS=10.0   SIGMA=3.3  Phase=50.0   Fom=0.4
INDEx 2 -3 1 FOBS=31.0   SIGMA=2.3  Phase=20.0   Fom=0.3
INDEx 5 6 6  FOBS=44.0   SIGMA=2.0
```

12.4.3 Example: Merging Crystallographic Reflection Files

The xrefin write statement allows one to merge reflection information. However, no relative scaling is performed. The following example shows how information can be merged. Suppose that reflection file "r.1" contains the following information:

```
index 1 2 3 fobs 40. phase 90. fom 0.5
index 2 3 4 fobs 30. phase 60. fom 0.4  weight 1.4
index 3 4 5 fobs 20. phase 50. fom 0.3
index -1 -2 -3 fobs 40. phase 90. fom 0.5   sigma 4.0
index -2 -3 -4 fobs 30. phase 60. fom 0.4
index -3 -4 -5 fobs 20. phase 50. fom 0.3
```

and reflection file "r.2" contains

```
index 1 2 3   fcalc 3. 3.5 fpart 4. 4.5 sigma 3.0
index -1 2 -5 fobs 20. fcalc 4. 4.5 weight 3.0
```

Merging of the two files can then be accomplished by using the following statements:

```
xrefin
```

```
  nreflection=100
  reflection @r.1 end
  reflection @r.2 end
  write reflection fobs fcalc output=merge.dat end
end
```

The resulting file "merge.dat" will contain the following information:

```
index 1 2 3 fobs 40. phase 90. fom 0.5  fcalc 3. 3.5 fpart 4. 4.5 sigma 3.0
index 2 3 4 fobs 30. phase 60. fom 0.4  weight 1.4
index 3 4 5 fobs 20. phase 50. fom 0.3
index -1 -2 -3 fobs 40. phase 90. fom 0.5   sigma 4.0
index -2 -3 -4 fobs 30. phase 60. fom 0.4
index -3 -4 -5 fobs 20. phase 50. fom 0.3
index -1 2 -5 fobs 20. fcalc 4. 4.5 weight 3.0
```

Note that reflection 1,2,3 is already present in the first file. Information has been merged for this reflection.

12.5 Manipulating Reflection Data

The DO statement allows one to carry out manipulations on F_{calc}, F_{obs}, $F_{partial}$, and various other arrays. It can also be used to define a weighting scheme. All operations are carried out for the selected reflections. The selection is accomplished by the xrefin FWINdow and RESOlution statements. The GENErate, EXPAnd, and REDUce statements can be used to complement data, expand the data to P_1, or reduce the data to an asymmetric unit. For a brief description of the latter three statements, see Section 12.3.

12.5.1 Syntax

DO <**xrefin-do-statement**> is invoked as an xrefin statement. Action takes place as soon as the construct is complete.

<**xrefin-do-statement**>:== <**xrefin-do-mode**> (<**xrefin-property**> = <**xrefin-do-expression**>)

<**xrefin-do-mode**>:==

AMPLitude carries out an operation using amplitudes of xrefin properties.

COMPlex carries out an operation in complex space.

PHASe carries out an operation using phases of xrefin properties. Phases are manipulated in degrees.

SCALe involves an operation that multiplies the left-hand side (lhs) by a scale factor k, which is defined as

$$k = \sum_{\vec{h}} W_{\vec{h}} |\mathrm{rhs}(\vec{h})||\mathrm{lhs}(\vec{h})| / (\sum_{\vec{h}} W_{\vec{h}} \mathrm{lhs}(\vec{h})^2) \tag{12.12}$$

where rhs represents the right-hand side of the equation. The action of this statement is identical to the scaling carried out when computing the R value or the residual (cf. Eqs. 12.1, 12.6), except that it refers the scaling to the right-hand side of the equation, whereas in the residual or R value evaluation the scaling refers to $F_c = F_{calc} + F_{part}$. Therefore, if the right-hand side of the scaling operation consists of F_{calc}, the program refers only to F_{calc} but not to to $F_c = F_{calc} + F_{part}$.

<xrefin-do-expression>:== **<xrefin-vflc>** [**<op>** **<xrefin-do-expression>**] represents a mathematical operation and is defined in Section 2.16.

<xrefin-vflc>:==

<function> | **<xrefin-function>** | **<symbol>** | **<real>** | **<integer>** | **<complex>** | **<xrefin-property>** represents mathematical functions and is defined in Section 2.16. Special xrefin functions are defined below. The data type of the function arguments has to match the data type of the operands.

<xrefin-property>:==

FCALC specifies calculated structure factors F_{calc} (complex).

FOBS specifies observed structure factors F_{obs} (complex).

FPART specifies partial structure factors F_{part} (complex).

SIGMA specifies σ values (real).

TEST is an array for cross-validation that allows one to compute the free R value (see Chapter 15).

WEIGHT specifies $W_{\vec{h}}$ weights (real).

<xrefin-function>:==

H() defines an array that consists of the Miller index h.

K() defines an array that consists of the Miller index k.

L() defines an array that consists of the Miller index l.

S() defines an array that consists of the length of the reciprocal lattice vector for each reflection h,k,l.

12.5.2 Requirements

The reflection data, unit cell parameters, and limits for the selection of reflections have to be defined.

12.5.3 Example: Scaling

In the following example, one adds F_{part} to F_{calc}, scales F_{calc} to the same magnitude as F_{obs}, and then computes $2F_{obs} - F_{calc}$ amplitudes. The phases of F_{calc} will not be touched. The operations are carried out for the selected reflections only. The operation involving F_{part} is necessary only when F_{part} is actually used; note that the do statement does not automatically add F_{part} to F_{calc}.

```
RESOlution 8. 2.5
FWINdow 0.1 100000.
DO COMPLEX (FCALC=FCALC+FPART)
DO COMPLEX (FPART=0)
DO SCALE (FCALC=FOBS)
DO AMPLITUDE (FCALC=2*FOBS-FCALC)
```

12.5.4 Example: Definition of a Weighting Scheme

This example shows how to define an arbitrary weighting scheme that is a function of SIGMA, FOBS, FCALC, S, H, K, and L. The statement should be inserted at any place after reading the reflections and before calculating the structure factors with the new weights. The following lines are examples of possible weighting schemes:

```
RESOlution 8. 2.5
FWINdow 0.1 100000.
DO (WEIGHT = 1. / SIGMA^2 )
DO (WEIGHT = 1. / ( 10. + 60. * ( S()/2. - 0.16667 ))^2 )
```

12.5.5 Example: Application of a 2σ Cutoff

The following example sets the F_{obs} to zero for reflections that have F_{obs} values less than 2σ. The FWINdow specification makes sure that all reflections with $F_{obs} = 0$ are excluded.

```
RESOlution 8. 2.5
FWINdow 0.1 100000.
DO AMPLITUDE (FOBS = FOBS * STEP(FOBS-2.0*SIGMA) )
```

12.5.6 Example: Generate a Full F_{calc} Data Set

In the following example, it is assumed that the atoms, atomic-form factors, and symmetry operators are well defined. GENErate sets all newly created F_{obs} to one, except for systematic absences, which are indicated by a zero value. The following example also works if there are reflections already present. In this case, GENErate complements the existing reflections so that a full asymmetric unit within the specified resolution limits is produced.

```
FWINDow 0.1 100000.
RESOlution 8. 2.
```

```
GENErate
UPDAte
WRITe REFLection FCALc END
```

12.5.7 Example: Expand a Data Set

In the following example, the current data set is expanded by application of the two crystallographic symmetry operators and by application of the Hermitian symmetry operator (i.e., computation of the Friedel mate). Only selected reflections are expanded. The resulting data set comprises a full sphere of data. After the operation, one has to turn off the symmetry operators explicitly.

```
SYMMetry=( x, y, z)
SYMMetry=( -x, y+1/2, -z)
HERMitian=TRUE

RESOlution 8. 2.
FWINdow 0.1 100000.

EXPAnd
SYMMetry RESEt
HERMitian=FALSE
```

To produce a hemisphere of data in P_1, use the following sequence of statements:

```
SYMMetry=( x, y, z)
SYMMetry=( -x, y+1/2, -z)

RESOlution 8. 2.
FWINdow 0.1 100000.

HERMitian=FALSE
EXPAnd
SYMMetry RESEt
HERMitian=TRUE
REDUce
```

12.6 Partial Structure Factors

Suppose that one wants to freeze a major portion of the stucture during refinement. Under certain conditions, one can save CPU time by storing the calculated structure factors of the frozen part of the molecule and then adding them to the calculated structure factors of the free part of the molecule. The computation of R values, crystallographic target functions, or electron density maps always refers to $F_c = F_{calc} + F_{part}$ (cf. Chapter 12). Note, however, that F_{part} is *not* added to F_{calc} when performing XREFin DO operations (Section 12.5).

In the following example, the structure factors of residues 1 and 2 are computed and written along with F_{obs} to the file "partial.data":

```
xrefin
  ...
  reflection @fobs.fob end
  selection=( resid 1:2 )
  update
  do (FPART=FCALC)
  write reflection fobs fpart output=partial.data end
end
```

In the next example, the partial structure factors are read, the structure factors of the remaining residues are computed, and R for whole structure is computed:

```
xrefin
  ...
  reflection reset @partial.data end
  selection=( not ( resid 1:2 ) )
  update
  print rfactor
end
```

In addition, one should fix the frozen atoms during minimization and dynamics by including the statement

```
constraints fix=( resid 1:2 ) end
```

before running minimization or dynamics. One should use "method=direct" instead of "method=FFT" if the frozen part contains more than 80% of the molecule.

12.7 Bulk Solvent Mask

A bulk solvent mask calculation is available in X-PLOR. The correction works by defining the solvent region in the unit cell of the crystal, assigning an average electron density to it, and Fourier-transforming this region. The $F_{solvent}$ terms are stored in the FPART array and added to the calculated structure factors (in FCALC) from the model being refined. The FFT method is always used for this calculation.

The solvent region is defined on the grid used for the model electron density generation for the FFT, which is initialized to the bulk solvent level at each point. By default, the bulk solvent density is 1.0 e^- $Å^{-3}$ to simplify scaling (see below). For each model atom, all grid points lying within the sum of the half van der Waals radii of the model atoms and an input solvent radius are set to zero. After passing through all model atoms, the remaining nonzero points define the solvent region. This procedure defines the accessible surface area of the centers of the water oxygen atoms. It is often useful to "shrink" this surface area by about half the van der Waals radius of the water oxygen atom. This shrinking procedure will produce a more realisitic estimate of the solvent volume and it is also useful for bulk solvent models.

12.7.1 Syntax

SOLMask { <**solmask-statement**> } END is invoked from the xrefin level. Action is taken after the END statement.

<**solmask-statement**>:==

SOLDen=<real> specifies the bulk solvent density in e^- $Å^{-3}$ (default: 1.0).

SOLRad=<real> specifies the solvent half-radius in Å (default: 1.4 Å).

SHRINk=<real> specifies the amount by which the mask is shrunken into the macromolecular region (default: 0.0 Å).

12.7.2 Requirements

The molecular structure, atomic coordinates, atomic-form factors, reflections, unit cell, and symmetry operators have to be known.

12.7.3 Example: Use of a Solvent Mask for Bulk Solvent Correction

Since the bulk solvent density is not usually known, the solvent structure factors must be scaled to the FCALC terms. Also, the grid being transformed has a sharp edge between the solvent and solvent-excluded regions, so Fourier ripples are generated that adversely affect the high-resolution terms. A large temperature factor can be applied to smooth these edge effects. Thus, a DO statement, FPART = FPART $k_{solvent}$ $exp(-B_{solvent}s^2/4)$, is used to correct the calculated solvent structure factors in FPART. $k_{solvent}$ and $B_{solvent}$ are determined empirically by iteratively searching for the value of one parameter that minimizes the R value in the lowest-resolution shell without significantly increasing the high-resolution R values, keeping the other parameter fixed. Typically, initial values of k=0.40 e^- $Å^{-2}$ and B=200 $Å^2$ are chosen. The X-PLOR shell language is used to loop over values of k in a coarse search; the optimum value of k is subsequently used in a coarse search to optimize B. This procedure is subsequently repeated over a finer set of values for k and B. All search procedures have to be done manually.

The following example shows how to compute the solvent mask:

```
 remarks  file xtalrefine/solmask.inp
 remarks  Compute solvent mask, do an FFT, and write out Fs

{===>} parameter @TOPPAR:parhcsdx.pro end                {*Read parameters.*}

{===>} structure @../generate/generate.psf end       {*Read structure file.*}

{===>} coor @slowcool.pdb                               {*Read coordinates.*}


 xrefine                                    {*This invokes the              *}
```

```
                                                {*crystallographic data parser.*}
{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                             {*Symmetry operators for space *}
   symmetry=(-x+1/2,-y,z+1/2)                   {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)                   {*Int. Tables.                 *}
   symmetry=(x+1/2,-y+1/2,-z)

   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=15000
   reflection @amy.fob end                                {*Read reflections.*}

{===>}
   resolution  40. 2.0        {*Resolution range, including low-resolution.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000

   method=FFT

   fft
      memory=1000000
   end

   update                          {*Update Fcalcs and print current rfactor.*}
   mbins=45
   print rfactor

 end


 xrefine

   solmask                                            {*Compute solvent mask.*}
     soldensity=1.0    {*Probe radius is 1.4 A; solvent density is 1 e/A^3.*}
     solrad=1.4
   end                     {*The FFT of the solvent mask is stored in FPART.*}


{===>}          {*Write a new reflection file including the solvent FPART.*}
   write reflection fobs sigma fpart    output=amy_s.fob end

 end

 stop
```

The next example shows how to test various scale and B-factors. Note that this protocol has to be run several times with different trial scale factors and B-factors. One should look at low R values for the low-resolution bins while maintaining low R values for the high-resolution bins. This is a

somewhat subjective procedure, which is the reason why this protocol is not automated.

```
 remarks  file xtalrefine/solscale.inp
 remarks  Determine scale factor and b-factor for solvent
 remarks  Fs by trial and error

{===>} parameter @TOPPAR:parhcsdx.pro end                {*Read parameters.*}

{===>} structure @../generate/generate.psf end       {*Read structure file.*}

{===>} coor @slowcool.pdb                               {*Read coordinates.*}


 xrefine                                        {*This invokes the             *}
                                                {*crystallographic data parser.*}
{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                             {*Symmetry operators for space *}
   symmetry=(-x+1/2,-y,z+1/2)                   {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)                   {*Int. Tables.                 *}
   symmetry=(x+1/2,-y+1/2,-z)

   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=15000
   reflection @amy_s.fob end      {*Read reflections, including solvent Fs.*}

{===>}
   resolution  40. 2.0        {*Resolution range, including low-resolution.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))   {*Sigma cut off.*}
   fwind=0.1=100000

   method=FFT

   fft
      memory=1000000
   end

 end

xrefine
   update                                                  {*Compute Fcalcs.*}

   mbins 45                           {*Number of bins for rfactor printout.*}

                         {*First loop keeps the B-factor fixed and varies k.*}
{===>}
   evaluate ($bfactor=100.)              {*Apply B of 200 when modifying k.*}

   do (FPART=( FPART*exp(-$bfactor*(s()^2)/4.) ))

                                        {*Determine scale factor k*FPART.*}
```

```
   evaluate ($1 = 0.390) {*Loop over k, starting at 0.39 and ending at 0.1.*}
   evaluate ($2 = 1.0)
   while ($1 > .100 ) loop solk
      evaluate ( $1 = $1 - .02 )
      do ( FPART = ($1/$2)*FPART)
      print rfactor
      evaluate ($2 = $1)
   end loop solk

   do ( FPART = (exp($bfactor*(s()^2)/4.) / $2)*FPART )
                                                  {*Retrieve original FPART.*}

                    {*Second loop keeps k constant and varies the B-factor.*}

{===>}
   evaluate ($kpart=0.31)  {*Apply a k of 0.27 when modifying the B-factor.*}

   do (FPART =  FPART*$kpart )

   evaluate ($1 = 200.)
   evaluate ($2 = 0.)
   while ($1 > 0. ) loop bfac
      evaluate ( $1 = $1 - 25.)
      do (FPART=( FPART*exp(-($1 - $2)*(s()^2)/4.) ))
      print rfactor
      evaluate ($2 = $1)
   end loop bfac

   do ( FPART = (exp($2*(s()^2)/4.) / $kpart)*FPART )
                                                  {*Retrieve original FPART.*}

                          {*Do a test with the current $kpart and $bfactor.*}
   do ( FPART= $kpart * exp(-$bfactor*(s()^2)/4.)*FPART )
   print rfactor
end

         {*Modify $kpart and $bfactor and rerun this job until the results*}
         {*are optimal when modifying either k or b.                      *}

stop
```

In subsequent protocols, one should include the following lines before computing R values, crystallographic targets, or electron density maps:

```
reflection @amy_s end            {*Reflection file with solvent mask FPART.*}
resolution 40. 2.
do ( FPART= 0.4 * exp(-200*(s()^2)/4.)*FPART )
```

12.8 Alternate Conformations

The following examples show how to set up alternate conformations for structure factor calculations as well as for empirical energy calculations.

The first step is to generate a molecular structure file that includes alternate conformations. The following example shows how to modify the "generate.inp" file (Section 3.13) in order to set up an alternate conformation involving residue 621.

```
remarks  file  xtalrefine/alternate.inp
remarks  Make alternate conformations

```

```
                                {* Read parameter and topology files as usual.*}

segment

  name="LY  "

  chain
    @TOPPAR:toph19.pep

             {* Coordinate file contains only one conformer with LY segid. *}
    coordinates @LY.PDB end
  end
end

                                                 {* Rename atoms if required. *}
vector do (name="O") ( name OT1 )
vector do (name="OT") ( name OT2 )
vector do (name="CD1") ( name CD and resname ile )

                               {* Here we establish the alternate conformers. *}
vector do (segid="LY-1") (resid 621 and not
          (name ca or name c or name n or name o or name cb or name h ))

duplicate select=( segid "LY-1" ) segid="LY-2" end

                                    {* Here we actually read the coordinates. *}
                 {* Note different file from that read to obtain sequence. *}
coordinates @LY-ALL.PDB end

   {* Now notify program that the alternate conformers can interact with   *}
   {* themselves or the rest of the protein, but not each other.           *}
   {* This information needed in other procedures, eg slowcool.inp as well.*}
constraints
  inter = (segid = "LY  ") (segid = "LY  ")
  inter = (segid = "LY  " or segid = "LY-1") (segid = "LY-1")
  inter = (segid = "LY  " or segid = "LY-2") (segid = "LY-2")
end

     {* Perform hydrogen building as usual.  Write molecular structure and *}
     {* coordinate file.                                                    *}
```

In subsequent protocols, e.g. "slowcool.inp", one has to insert the following statement after reading the molecular structure file:

```
constraints
  inter = (segid = "LY  ") (segid = "LY  ")
  inter = (segid = "LY  " or segid = "LY-1") (segid = "LY-1")
  inter = (segid = "LY  " or segid = "LY-2") (segid = "LY-2")
end
```

The constraints interaction statements make sure that the alternate conformations do not interact with each other but rather interact only with the remaining molecular structure.

The occupancies of the alternate conformations can be read from the coordinate file or explicitly set by using

```
vector do (q=0.5) ( segid "LY-1" )
vector do (q=0.5) ( segid "LY-2" )
```

Now one is ready to carry out structure factor calculations as well as empirical energy calculations. In order to refine occupancies, one should use the xrefin optimize group statement.

```
xrefin
   optimize group
      q=( segid "LY-1" and resid 621 )
      q=( segid "LY-2" and resid 621 )
       nstep=15
       drop=1
    end
end
```

Note that at present the occupancy refinement procedure does not constrain the sum of occupancies to be one. Manual resetting of the occupancies may be required.

12.9 Anomalous Scattering

X-PLOR allows one to incorporate the effects of anomalous scattering into the structure factor calculation. There are three places where input files have to be modified: including the imaginary constant for specification of the appropriate atomic-form factor, specifying the ASELection statement, and turning off the Hermitian symmetry operation.

The following example illustrates the inclusion of anomalous scattering effects for FE atoms:

```
 remarks  file  xtalrefine/anomalous.inp  -- Example of how
 remarks  to deal with anomalous scattering

{===>} parameter @TOPPAR:parhcsdx.pro end                {*Read parameters.*}

{===>} structure @../generate/generate.psf end       {*Read structure file.*}

{===>} coor @prepstage.pdb                              {*Read coordinates.*}

 vector do ( charge=0.0 ) ( resname LYS and
    ( name ce or name nz or name hz* ) )         {*Turn off charges on LYS.*}
 vector do ( charge=0.0 ) ( resname GLU and
    ( name cg or name cd or name oe* ) )         {*Turn off charges on GLU.*}
 vector do ( charge=0.0 ) ( resname ASP and
    ( name cb or name cg or name od* ) )         {*Turn off charges on ASP.*}
 vector do ( charge=0.0 ) ( resname ARG and
    ( name cd or name *E or name cz or name NH* or name HH* ) )
                                                 {*Turn off charges on ARG.*}

 flags
    include pele pvdw xref
    ?
 end


 xrefine

{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                          {*Symmetry operators for space *}
   symmetry=(-x+1/2,-y,z+1/2)                {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)                {*Int. Tables.                 *}
   symmetry=(x+1/2,-y+1/2,-z)

                                                        {*Normal scatterers.*}
   SCATter ( chemical C* )
```

```
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490

{===>}                                          {*Anomalous scatterers.*}
   SCATter ( chemical FE* )                                {* For FE+3. *}
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566
          {* Anomalous dispersion corrections are for Cr Kalpha radiation. *}
           -0.3683                      {* This is the sum of c and f'. *}
           IMAGinary  0.764                            {*This is f''. *}

   aselection=( chemical FE* )
   anomalous=true                             {*Turn off Friedel symmetry.*}

{===>}
   nreflections=15000
   reflection @amy.fob end     {*Read reflections, including Friedel mates.*}

{===>}
   resolution  5.0 2.0                                {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000

   method=FFT

   fft
      memory=1000000
   end

   update

   print rfactor

end

           {*Now one is ready for minimization, dynamics, map calculation,*}
           {*and so forth.                                                 *}

stop

```

12.10 Special Positions

Special positions are recognized automatically by X-PLOR. In particular, the occupancy of a special position by an atom should not be scaled, since X-PLOR takes care of this automatically. In order for the program to recognize the special position, the atomic coordinate has to be exact. Note that covalent bonds to atoms at special positions are not possible at present.

During positional refinement atoms at special positions need to be fixed by using the CONStraints FIX statement (Section 8.1). Otherwise, an atom might move away from the special position due to round-off errors and cause a very large van der Waals repulsion.

Several users have noticed that it is not possible to define covalent bonds between an atom at a special position and other atoms without causing problems with the nonbonded energy term. A work-around consists of using the CONStraints INTEraction statement to explicitly exclude symmetry-related non-bonded interactions between the special atom and all other atoms by modifying the PVDW weight. The following example assumes that the name of the atom at the special position is "SP".

```
CONStraints
 INTEr (not name "SP") (not name "SP")   WEIGht PVDW 1 END
 INTEr (name "SP") (all)                 WEIGht PVDW 0 END
END

CONStraints  FIX=( name "SP" ) end
```

12.11 Inclusion of all Hydrogens

Normally, hydrogens cannot be resolved in macromolecular crystal structures. Therefore, the scattering produced by hydrogens is usually neglected. However, X-PLOR provides the possibility of including all hydrogens in the structure factor calculation as outlined in the following.

1. The molecular structure file needs to be generated with the all-hydrogen set. Use an all-hydrogen parameter and topology set such as "topallhdg.pro" and "parallhdg.pro".

2. The bond length between hydrogen and other atoms needs to be shorted to 0.5 Å in order to properly model the distribution of electron density. This can be accomplished by inserting the following statement just before the hbuild statement in the input file for structure file generation and in all subsequent input files right after reading the structure and parameter files:

```
parameter
   bonds ( all ) ( hydrogen ) 500. 0.5
end
```

3. The definition for the atomic form factor for hydrogens needs to be added to all input files where structure factors are calculated.

```
SCATter ( chemical H* )
 0.48992 20.6593 0.26200 7.74039 0.19677 49.5519 0.04988 2.20159 0.00131
```

The addition of all hydrogens significantly increases the CPU requirements. Furthermore, the short bond lengths involving hydrogens might cause numerical instabilities. Reduction of the timestep or temperature may be required in slowcooling protocols.

12.12 Luzzati Plot

The R value as a function of resolution and intensity can be obtained by issuing the PRINt R statement after structure factors have been computed. The output of the PRINt R statement can be used to make a plot of the R value distribution, the so-called Luzzati plot (Luzzati 1952). In the following example, the X-PLOR script file creates a file "luzzati.list":

```
remarks   file xtalrefine/luzzati.inp  -- Makes a Luzzati plot
remarks   R value as a function of 1/d, where d is the resolution

                                                    {*Read structure file.*}
{===>} structure @../generate/generate.psf end

{===>} coordinates @slowcool.pdb                       {*Read coordinates.*}

{===>}
evaluate ( $low_resolution= 10. )              {*Set the resolution limits.*}
evaluate ( $high_resolution= 2. )

xrefine

{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                          {*Symmetry operators for space *}
   symmetry=(-x+1/2,-y,z+1/2)                {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)                {*Int. Tables.                 *}
   symmetry=(x+1/2,-y+1/2,-z)

   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=15000
   reflection @amy.fob end                              {*Read reflections.*}

   resolution  $low_resolution   $high_resolution

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000


   method=FFT             {*Use the FFT method instead of direct summation.*}

   fft
      memory=1000000
   end
end


{===>} set print=luzzati.list end  {*Set the filename for the rfactor list.*}

xrefin
   mbins=20                       {*Number of bins for R value calculation.*}
```

```
60
61    resolution $high_resolution $low_resolution
62
63    update                                                {*Compute Fcalcs.*}
64
65    print R                              {*Output is written to the print file.*}
66 end
67
68
69 stop
```

The following Mathematica script can be used to make the Luzzati plot shown in Fig. 12.1:

```
1             (* file  xtalrefine/luzzati.math  -- Makes a Luzzati plot      *)
2             (* R-factor error curves are plotted for errors between       *)
3             (* 0.5 and 0.2 A (in 0.05 intervals).                         *)
4
5            (* The error table information is taken from Luzzati's paper.   *)
6
7 Off[General::spell1];
8 $DefaultFont={"Times-Roman",13};
9
10
11                                      (* Change the name of the list file. *)
12 (*===>*) dd= <<luzzati.list;
13
14   col1=Transpose[dd[[1]]][[1]];
15   col2=Transpose[dd[[1]]][[2]];
16   col4=Transpose[dd[[1]]][[4]];
17   rfact=Transpose[{2/(col1+col2),col4}];
18   min=Min[2/(col1+col2)];
19   max=Max[2/(col1+col2)];
20   Ltable={{0.,0.},{0.01,0.025},{0.02,0.050},{0.03,0.074},{0.04,0.098},
21           {0.05,0.122},{0.06,0.145},{0.07,0.168},{0.08,0.191},
22           {0.09,0.214},{0.1,0.237},{0.12,0.281},{0.14,0.319},
23           {0.16,0.353},{0.18,0.385},{0.20,0.414},{0.22,0.44},
24           {0.24,0.463},{0.26,0.483},{0.28,0.502},{0.30,0.518},
25           {0.35,0.548},{0.40,0.564},{0.45,0.574},{0.50,0.580}};
26   coll1=Transpose[Ltable][[1]];
27   coll2=Transpose[Ltable][[2]];
28   Show[Graphics[{ Line[rfact],
29               GrayLevel[0.5],
30               Line[Transpose[{coll1/0.50,coll2}]],
31               Line[Transpose[{coll1/0.45,coll2}]],
32               Line[Transpose[{coll1/0.40,coll2}]],
33               Line[Transpose[{coll1/0.35,coll2}]],
34               Line[Transpose[{coll1/0.30,coll2}]],
35               Line[Transpose[{coll1/0.25,coll2}]],
36               Line[Transpose[{coll1/0.20,coll2}]]},
37   Ticks->{Range[min ,
38          max,0.05],Range[0.,0.6,0.05]},
39   Axes->True,
40   AxesOrigin->{min ,0.},AspectRatio->1,
41   PlotRange->
42     {{min  ,max +0.05 },{0.,0.7}},
43   AxesLabel->{"1/d","R factor"},
44   PlotLabel->"Luzzati Plot (0.5 to 0.2 )" ]];
45
```

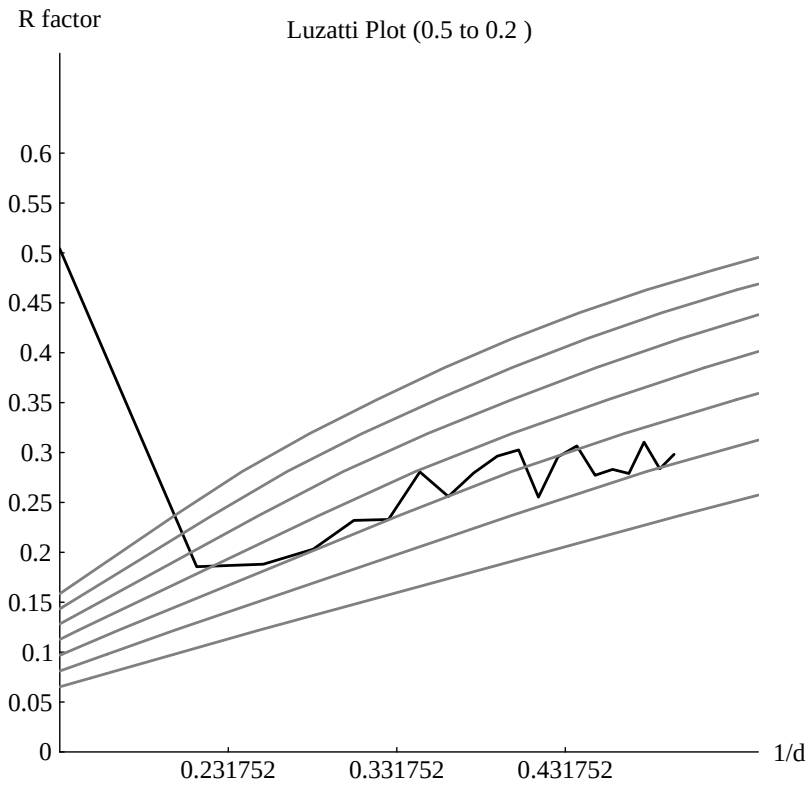

Figure 12.1: Luzzati plot.

12.13 Wilson Plot

The following example shows how to produce a Wilson plot (Wilson 1949; Rogers 1965; Main 1975). The X-PLOR script file creates a file "wilson.list":

```
remarks file xtalrefine/wilson.inp
remarks  Wilson plot.  If no molecular structure present see note at the
remarks  bottom

                                          {*Read structure file.          *}
{===>} structure @../generate/generate.psf end

{===>} coordinates @slowcool.pdb          {* Read coordinates.            *}
                        {* Note: the Wilson plot is independent of atomic  *}
                        {* positions, but X-PLOR produces warning messages *}
                        {* if atomic coordinates are unknown.  Just read   *}
                        {* any coordinate set or define fake coordinates.  *}




set print=wilson.list end

xrefine

{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

```

```
{===>}
   symmetry=(x,y,z)                          {*Symmetry operators for space   *}
   symmetry=(-x+1/2,-y,z+1/2)                {*group P212121 notation as in   *}
   symmetry=(-x,y+1/2,-z+1/2)                {*Int. Tables.                   *}
   symmetry=(x+1/2,-y+1/2,-z)

   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=15000
   reflection @amy.fob end                                {*Read reflections.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000


   method=FFT        {*Use the FFT method instead of direct summation.     *}

   fft
      memory=1000000
   end

   mbins=40

   resolution=10.0 2.

   print wilson
end


stop

           {* If no molecular structure is present, simply generate a fake *}
           {* structure with the appropriate number of point atoms. Use the*}
           {* following statements instead of the structure and coordinate *}
           {* statements.                                                  *}

!topology
!    residue C
!      atom=C type C mass=12.0 end
!    end
!    residue N
!      atom=N type N mass=14.0 end
!    end
!    residue O
!      atom=O type O mass=16.0 end
!    end
!    residue S
!      atom=S type S mass=32.0 end
!    end
!end
!
!segment                              {* Insert the actual number of atoms. *}
!    mole name=C number=345 end
!    mole name=N number=93  end
!    mole name=O number=116  end
!    mole name=S number=4   end
!end
```

```
!
!vector do (x=1.) ( all )                    {* Generate fake coordinates *}
!vector do (y=1.) ( all )                    {* to avoid error messages   *}
!vector do (z=1.) ( all )                    {* about unknown atoms.      *}
```

The following Mathematica script can be used to make the Wilson plot shown in Fig. 12.2:

```
             (* file  xtalrefine/wilson.math  -- Makes a wilson plot      *)
Off[General::spell1];
$DefaultFont={"Times-Roman",13};

                                      (* Change the name of the list file. *)
(*===>*) dd= << wilson.list;

  col1=Transpose[dd[[1]]][[3]];
  col7=Transpose[dd[[1]]][[7]];
  col8=Transpose[dd[[1]]][[8]];
  logff=Transpose[{col1,col7}];
  lsline=Transpose[{col1,col8}];
  smin=Min[col1];
  smax=Max[col1];
  sdelta=Round[(smax-smin)*100.0-0.5]/1000.0;
  lmin=Min[col7];
  lmax=Max[col7];
  ldelta=Round[(lmax-lmin)*100.0-0.5]/1000.0;
  pts = Table[Point[logff[[i]]],{i,Length[col7]}];
  Show[ Graphics[ { PointSize[0.01], pts,
                    GrayLevel[0.5],
                    Line[lsline]},
  AxesOrigin->{0.0,lmin-0.5}, AspectRatio->0.6,
  Axes->True,
  Ticks->{Range[0.0,smax+sdelta,3*sdelta],
          Range[lmin-0.5,lmax+0.5,3*ldelta]},
  PlotRange->{{0.0,smax+sdelta },{lmin-0.5,lmax+0.5}},
  AxesLabel->{"S^2","Log(<FF>/<ff>)"},
  PlotLabel->"Wilson Plot" ] ];
```

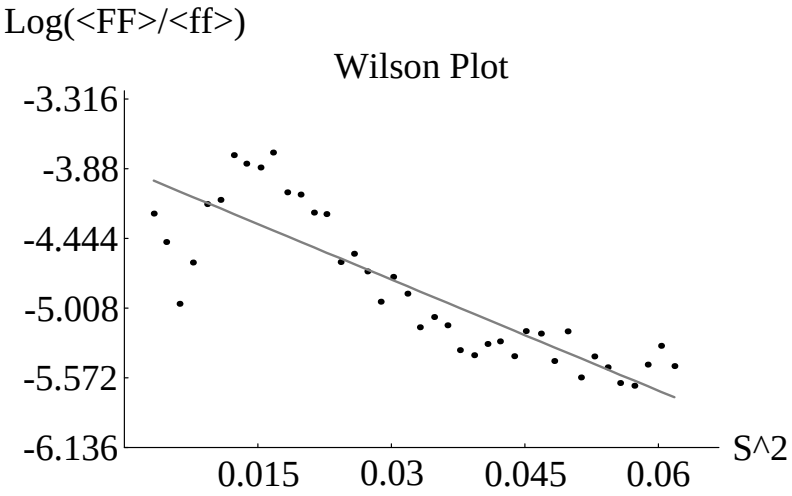

Figure 12.2: Wilson plot.

13 Crystallographic Refinement

The following examples show how to carry out various types of crystallographic refinement. They include positional refinement as well as several types of B-factor and occupancy refinements.

For general review of refinement techniques, the reader is referred to Stout and Jensen (1989) and Hendrickson (1985). For more information about simulated annealing refinement, the following articles are suggested: Brünger (1988), Brünger (1991a), Brünger (1991b), Brünger, Karplus, and Petsko (1989), Brünger, Krukowski, and Erickson (1990), Brünger, Kuriyan, and Karplus (1987), and Weis et al. (1990).

Figure 13.1 provides an overview of the crystallographic refinement features in X-PLOR. The boxes enclosed in black lines indicate external input (diffraction data, initial atomic model) or manipulations using "FRODO," "O," or other crystallographic modeling packages.

To generate the molecular structure, see Section 3.7. The following parameter and topology files are particularly useful for crystallographic refinement of proteins: "tophcsdx.pro" and "parhcsdx.pro" (Section 3.6). If after inspection of the model and electron density maps new atoms have to be added to the model, the molecular structure has to be appended or recreated.

13.1 Positional Refinement

The following examples illustrate how positional refinement can be carried out using X-PLOR. Both conventional refinement (conjugate gradient minimization) and refinement by simulated annealing are discussed.

13.1.1 Check of Data and Initial Structure and Determination of Weights

This procedure checks the initial R value and determines the ideal weight W_A (and W_P if a phase restraint is used) between E_{XREF} and $E_{EMPIRICAL}$. The procedure should be carried out before running any refinement on the

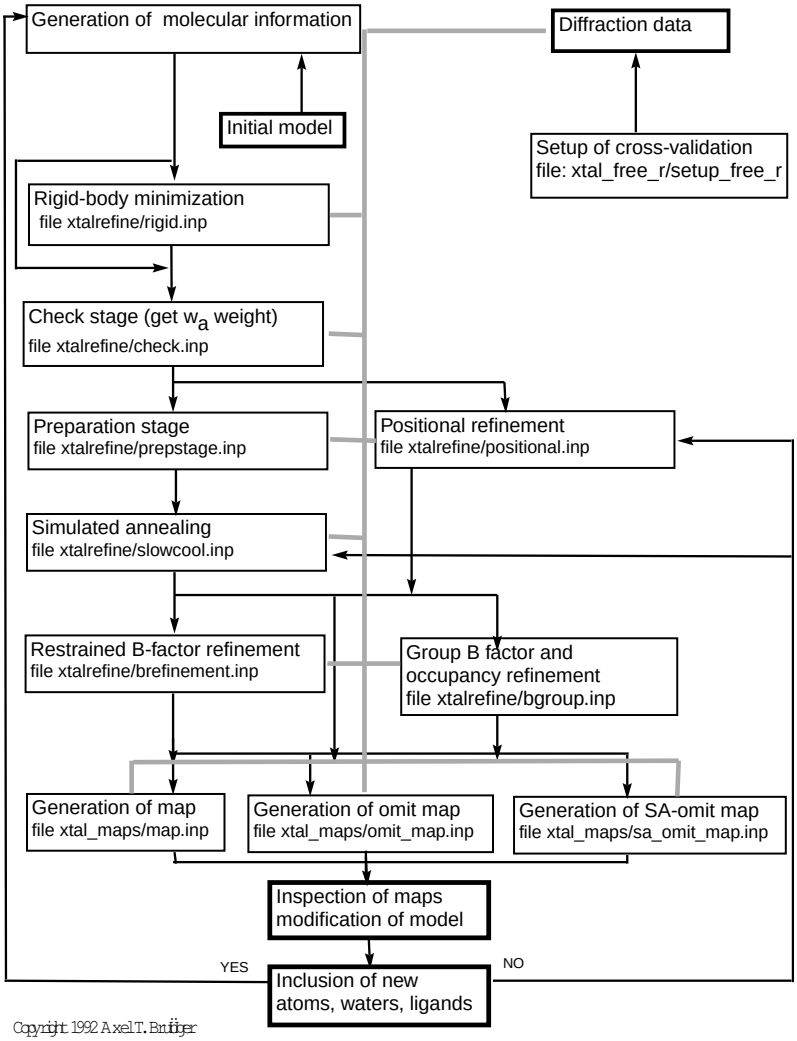

Figure 13.1: Overview of crystallographic refinement.

system. The weight W_A is determined by running a brief molecular dynamics calculation without E_{XREF} and then comparing the norm of the gradient of $E_{EMPIRICAL}$ and the norm of the gradient of E_{XREF}. Experience has shown that this ratio has to be scaled by a factor of about 1/2 to 1/3 in order to maximize the model's phase accuracy. The program will write this scaled weight to the output file (look for the line marked "display"). Refinement is not sensitive to changes in the weight of up to 25%. The scaled weight should be alright for most cases. If in doubt, the free R factor should be used to optimize the weight (see Section 15). A similar procedure can be applied when using targets other than the crystallographic residual (Eq. 12.1).

```
1  remarks  file  xtalrefine/check.inp
2  remarks  Check initial R value,
3  remarks  compute ideal weights for diffraction energy terms
```

```

{===>} parameter @TOPPAR:parhcsdx.pro end                {*Read parameters.*}

{===>} structure @../generate/generate.psf end       {*Read structure file.*}

{===>} coor @../generate/generate.pdb                   {*Read coordinates.*}

{===>}                                      {*If the temperature factors and*}
 vector do (b=15.0) ( all )                 {*occupancies are not defined in*}
 vector do (q=1.0) ( all )                  {*the coordinate file,          *}
                                            {*set them now.                 *}

 vector do ( charge=0.0 ) ( resname LYS and
    ( name ce or name nz or name hz* ) )         {*Turn off charges on LYS.*}
 vector do ( charge=0.0 ) ( resname GLU and
    ( name cg or name cd or name oe* ) )         {*Turn off charges on GLU.*}
 vector do ( charge=0.0 ) ( resname ASP and
    ( name cb or name cg or name od* ) )         {*Turn off charges on ASP.*}
 vector do ( charge=0.0 ) ( resname ARG and
    ( name cd or name *E or name cz or name NH* or name HH* ) )
                                                 {*Turn off charges on ARG.*}

 flags                                      {*In addition to the empirical  *}
    include pele pvdw xref                  {*potential energy terms, which *}
    ?                                       {*are turned on initially,  this*}
 end                                        {*statement turns on the        *}
                                            {*crystallographic residual term*}
                                            {*and packing term.             *}

 xrefine                                     {*This invokes the             *}
                                             {*crystallographic data parser.*}
{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                          {*Symmetry operators for space *}
   symmetry=(-x+1/2,-y,z+1/2)                {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)                {*Int. Tables.                 *}
   symmetry=(x+1/2,-y+1/2,-z)
            {*The following contains the atomic form factors.  A 4-Gaussian*}
            {*approximation is used.  Atoms are selected based on their    *}
            {*chemical type.  Note the use of wildcards in the selection.  *}

   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=15000
   reflection @amy.fob end                                {*Read reflections.*}

{===>}
   resolution  5.0 2.0                                    {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000

   print completeness                          {*Check completeness of data.*}

   method=FFT             {*Use the FFT method instead of direct summation.*}
```

```
74
75    fft
76       memory=1000000  {*This tells the FFT routine how much physical memory*}
77    end                {*is available; the number refers to DOUBLE COMPLEX  *}
78                       {*words, the memory is allocated from the HEAP.      *}
79
80
81    ?                                       {*This prints the current status.*}
82
83  end                          {*This terminates the diffraction data parser.*}
84
85  xrefin                                        {*This statement computes the*}
86                                                {*initial R value.           *}
87                                                  {*Please check the R value.*}
88    update-fcalc                              {*This is the last check before*}
89    print R                                   {*wasting a lot of computer    *}
90  end                                         {*time.                        *}
91
92  flags                                         {*TURN OFF diffraction terms *}
93    exclude xref                                {*but KEEP packing term and  *}
94  end                                           {*empirical potential energy.*}
95
96  minimize powell                             {*Invoke the Powell minimizer.*}
97
98    nstep=40                                                {*Do 40 cycles.*}
99
100   drop=40.0                                 {*This is the expected initial*}
101                                             {*drop in energy; the value is*}
102                                             {*not critical: 40.0 should be*}
103                                             {*reasonable in most cases.   *}
104
105 end                                            {*Minimization is executed.*}
106
107 dynamics  verlet                             {*Invoke Verlet integration.*}
108
109   timestep=0.001                              {*Time step in picoseconds.*}
110   nstep=100                                 {*Number of integration steps.*}
111
112   iasvel=maxwell                           {*Initial velocities from       *}
113   firsttemperature=300                     {*Maxwell distribution at 300 K.*}
114
115   ieqfrq=25                                   {*Scale the velocities every*}
116   finaltemperature=300                        {*25 steps.                  *}
117
118   nprint=50 iprfrq=100                                    {*Output control.*}
119   nsavc=0                                 {*Do not write a trajectory file.*}
120
121 end                                          {*Integration will be executed.*}
122
123                                            {*Now compute the ideal weight *}
124                                            {*at the specified resolution  *}
125                                            {*range by comparing gradients.*}
126 energy end
127 evaluate ($grad_ener=$grad)
128 xrefin
129   wa=1. wp=0.
130   resolution 5. 2.
131 end
132 flags exclude * include xref end
133 energy end
134             {* Experience has shown that the gradient ratio has to be    *}
135             {* divided by a factor of two to three for optimum Rfree and *}
136             {* maximum phase accuracy.                                   *}
137 evaluate ($weight= $grad_ener/(2.* $grad))
138
139 display  suggested weight wa = $weight
140
141           {*To determine the ideal weight for the phase-dependent term,*}
142           {*uncomment the following lines.                             *}
143 ! xrefin
```

```
!   wa=0. wp=1.
!   resolution 5. 2.
! end
! energy end
!             {* Experience has shown that the gradient ratio has to be    *}
!              {* divided by a factor of two to three for optimum Rfree and *}
!              {* maximum phase accuracy.                                    *}
!evaluate ($weight= $grad_ener/(2.* $grad))
!
! display  suggested weight wp = $weight

 stop
```

If the minimizer exits with "line search abandoned" and very large van der Waals energies (in the thousands of kcal mole^{-1}), the structure probably has bad contacts. The output of X-PLOR provides a list of bad contacts . One should check the molecule and symmetry-related molecules on the graphics and fix the problem. If this is inconvenient, try the following procedure: use the "repel" nonbonded energy function (see Section 4.1) for the first several steps of minimization. One should replace the minimization statement in the example above with the following statements:

```
 parameter nbonds repel=0.89 rcon=4. end end
                                              {*This turns on the repulsive *}
                                              {*function.                   *}
 minimize powell
   nstep=40
   drop=40.0
 end
                        {* This switches back to the Lennard-Jones function.*}
 parameter nbonds repel=0.0 end end

 minimize powell
   nstep=40
   drop=40.0
 end
```

13.1.2 Conventional Positional Refinement

This example shows how to carry out conventional refinement using a conjugate gradient minimization. The "check.inp" protocol should be carried out before running this protocol. The repel nonbonded energy function (Eq. 4.8) should be tried if the minimization does not converge or exits with "line search abandoned" and van der Waals energies in the thousands of kcal mole^{-1} (cf. Section 13.1.1).

```
 remarks  file  xtalrefine/positional.inp  -- Conventional positional
 remarks  refinement

{===>} parameter @TOPPAR:parhcsdx.pro end                {*Read parameters.*}

{===>} structure @../generate/generate.psf end       {*Read structure file.*}

{===>} coor @prepstage.pdb                              {*Read coordinates.*}

 vector do ( charge=0.0 ) ( resname LYS and
    ( name ce or name nz or name hz* ) )         {*Turn off charges on LYS.*}
 vector do ( charge=0.0 ) ( resname GLU and
```

```
    ( name cg or name cd or name oe* ) )         {*Turn off charges on GLU.*}
 vector do ( charge=0.0 ) ( resname ASP and
    ( name cb or name cg or name od* ) )         {*Turn off charges on ASP.*}
 vector do ( charge=0.0 ) ( resname ARG and
    ( name cd or name *E or name cz or name NH* or name HH* ) )
                                                 {*Turn off charges on ARG.*}

 flags
    include pele pvdw xref
    ?
 end


 xrefine

{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                          {*Symmetry operators for space *}
   symmetry=(-x+1/2,-y,z+1/2)                {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)                {*Int. Tables.                 *}
   symmetry=(x+1/2,-y+1/2,-z)

   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=15000
   reflection @amy.fob end                               {*Read reflections.*}

{===>}
   resolution  5.0 2.0                                   {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000

   method=FFT

   fft
      memory=1000000
   end


   tolerance=0.0 lookup=false             {*This makes the minimizer happy.*}

{===>}
   wa=26000                                  {*Weight from job "check.inp".*}
 end


 minimize powell                             {*Invoke the Powell minimizer.*}

   nstep=40                                                {*Do 40 cycles.*}

   drop=10.0                                 {*This is the expected initial*}
                                             {*drop in energy; the value is*}
                                             {*not critical: 10.0 should be*}
                                             {*reasonable in most cases.   *}
```

```

 end                                                {*Minimization is executed.*}

 write coordinates output=positional.pdb end      {*Write final coordinates.*}
 stop
```

13.1.3 Crystallographic Refinement by Simulated Annealing

The following two example input files demonstrate how to use simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983) through molecular dynamics in crystallographic refinement (SA-refinement). The "check.inp" protocol should be carried out before running this protocol.

The first step is the preparation stage of SA-refinement. It consists of several cycles of minimization in order to relieve strain or bad contacts of the initial coordinates. The repel nonbonded energy function (Eq. 4.8) should be tried if the minimization does not converge or exits with "line search abandoned" and van der Waals energies in the thousands of kcal mole^{-1} (cf. 13.1.1).

```
 remarks  file  xtalrefine/prepstage.inp
 remarks  Prepare structure for simulated annealing

{===>} parameter @TOPPAR:parhcsdx.pro end                {*Read parameters.*}

{===>} structure @../generate/generate.psf end       {*Read structure file.*}

{===>} coor @../generate/generate.pdb                   {*Read coordinates.*}

{===>}                                        {*If the temperature factors and*}
 vector do (b=15.0) ( all )                   {*occupancies are not defined in*}
 vector do (q=1.0) ( all )                    {*the coordinate file,          *}
                                              {*set them now.                 *}

 vector do ( charge=0.0 ) ( resname LYS and
    ( name ce or name nz or name hz* ) )          {*Turn off charges on LYS.*}
 vector do ( charge=0.0 ) ( resname GLU and
    ( name cg or name cd or name oe* ) )          {*Turn off charges on GLU.*}
 vector do ( charge=0.0 ) ( resname ASP and
    ( name cb or name cg or name od* ) )          {*Turn off charges on ASP.*}
 vector do ( charge=0.0 ) ( resname ARG and
    ( name cd or name *E or name cz or name NH* or name HH* ) )
                                                  {*Turn off charges on ARG.*}

 flags
    include pele pvdw xref
    ?
 end

 xrefine

{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                           {*Symmetry operators for space *}
   symmetry=(-x+1/2,-y,z+1/2)                 {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)                 {*Int. Tables.                  *}
   symmetry=(x+1/2,-y+1/2,-z)

   SCATter ( chemical C* )
```

```
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=15000
   reflection @amy.fob end                              {*Read reflections.*}

{===>}
   resolution  5.0 2.0                                  {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000

   method=FFT

   fft
      memory=1000000
   end


   tolerance=0.05                               {*Tolerance for the minimizer.*}

{===>}
   wa=26000                                     {*Weight from job "check.inp".*}
 end

 vector do (refx=x) ( all )                      {*The structure is restrained*}
 vector do (refy=y) ( all )                      {*to the main coordinate set *}
 vector do (refz=z) ( all )                      {*(generate.pdb).            *}

 vector do (harmonic=0.0) ( all )         {*Set the harmonic energy constant*}
 vector do (harmonic=20.0) ( name ca )    {*to 20.0 kcal/(mole A**2).       *}
 flags include harm end

 minimize powell                                {*Invoke the Powell minimizer.*}

   nstep=40                                    {*If the initial energy of the  *}
                                               {*structure is large, one should*}
                                               {*do at least 160 cycles.       *}

   drop=40.0                                    {*This is the expected initial *}
                                                {*drop in energy; this value is*}
                                                {*not critical: 40.0 should be *}
                                                {*reasonable in most cases.    *}

 end                                           {*Minimization will be executed.*}

 write coordinates output=prepstage.pdb end            {*Write coordinates.*}

 stop
```

Now we are ready for simulated annealing. The following example represents the so-called slow-cooling protocol (Brünger, Krukowski, and Erickson 1990). When higher initial temperatures are used, one has to decrease the time step; e.g., one should use a 0.5 fsec time step at 4000K.

```
 remarks  file  xtalrefine/slowcool.inp
 remarks  Crystallographic SA-refinement (slow-cooling method)

{===>} parameter @TOPPAR:parhcsdx.pro end                {*Read parameters.*}

{===>} structure @../generate/generate.psf end       {*Read structure file.*}

{===>} coor @prepstage.pdb                              {*Read coordinates.*}

 vector do ( charge=0.0 ) ( resname LYS and
    ( name ce or name nz or name hz* ) )          {*Turn off charges on LYS.*}
 vector do ( charge=0.0 ) ( resname GLU and
    ( name cg or name cd or name oe* ) )          {*Turn off charges on GLU.*}
 vector do ( charge=0.0 ) ( resname ASP and
    ( name cb or name cg or name od* ) )          {*Turn off charges on ASP.*}
 vector do ( charge=0.0 ) ( resname ARG and
    ( name cd or name *E or name cz or name NH* or name HH* ) )
                                                  {*Turn off charges on ARG.*}

 flags
    include pele pvdw xref
    ?
 end

 xrefine

{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                          {*Symmetry operators for space *}
   symmetry=(-x+1/2,-y,z+1/2)                {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)                {*Int. Tables.                 *}
   symmetry=(x+1/2,-y+1/2,-z)

   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=15000
   reflection @amy.fob end                              {*Read reflections.*}

{===>}
   resolution  5.0 2.0                                  {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000

   method=FFT

   fft
      memory=1000000
   end


   tolerance=0.2                                    {*Tolerance for dynamics.*}

{===>}
   wa=26000                                    {*Weight from job "check.inp".*}
```

```
 end


 set seed=432324368 end {*Set the initial random seed for the v-assignment.*}

{===>}
 evaluate ($init_temp=3000. )                          {*Starting temperature.*}
 vector do (vx=maxwell($init_temp)) ( all )
 vector do (vy=maxwell($init_temp)) ( all )
 vector do (vz=maxwell($init_temp)) ( all )

 vector do (fbeta=100.) ( all )

 evaluate ($1=$init_temp)
 while ($1 > 300.0) loop main

    dynamics  verlet
      timestep=0.0005
      nstep=50
      iasvel=current
      nprint=5  iprfrq=0
      tcoupling=true tbath=$1
    end
    evaluate ($1=$1-25)
 end loop main


 xrefin

   tolerance=0.0    lookup=false          {*This makes the minimizer happy.*}
 end

 minimize powell                                       {*Final minimization.*}
   nstep=120
   drop=10.0
 end

 write coordinates output=slowcool.pdb end             {*Write coordinates.*}

 stop
```

If the temperature gets too high and the system blows up, one should try to reduce the time step or decrease the temperature. The final coordinates ("slowcool.pdb") are already minimized, unlike earlier protocols, which required a final stage.

13.1.4 Rigid-Body Refinement

X-PLOR provides the possibility of refinement of several rigid groups (see Section 9.2). The following example file is an application to crystallographic refinement. Parts of the molecule that are not specified in any GROUp statement remain fixed. The constraints fix statement has no influence on rigid-body refinement. In the following example, the orientational and translational parameters of two domains (residues 1–30 and 31–70) of the molecule are refined against the crystallographic target function. Other energy terms, such as van der Waals repulsion, may be included to avoid "unphysical" configurations of the rigid bodies.

```
 remarks  file  xtalrefine/rigid.inp
 remarks  Rigid-body refinement against the XREF term

```

```
{===>} structure @../generate/generate.psf end       {*Read structure file.*}

{===>} coor @prepstage.pdb                             {*Read coordinates.*}

 flags
    exclude * include  xref   {* In this example we use the XREF term only.*}
    ?
 end


 xrefine

{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                          {*Symmetry operators for space *}
   symmetry=(-x+1/2,-y,z+1/2)                {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)                {*Int. Tables.                 *}
   symmetry=(x+1/2,-y+1/2,-z)

   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=15000
   reflection @amy.fob end                              {*Read reflections.*}

{===>}
   resolution  5.0 2.0                                  {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * heavy(fobs - 2.0*sigma))    {*Sigma cutoff.*}
   fwind=0.1=100000

   method=FFT

   fft
      memory=1000000
   end


   tolerance=0.0 lookup=false        {* These options are required for the *}
                                     {* minimizer.*}

   wa=10000                          {* Use this weight if the only active *}
                                     {* energy term is XREF, otherwise, use*}
                                     {* the weight obtained from the       *}
                                     {* "check.inp" protocol.              *}
 end


 minimize rigid

      nstep=40                                      {*Do 40-step rigid-body*}
                                                    {*minimization.        *}

      drop=10.0                           {*Expected initial drop in energy.*}

      group=( resid 1:30 )                        {*Select two rigid groups.*}
```

```
74       group=( resid 31:70 )
75
76   end
77
78   write coordinates output=rigid.pdb end          {*Write final coordinates.*}
79
80   stop
```

13.2 Overall B-Factor Refinement

The xrefin optimize overall statement optimizes overall isotropic or anisotropic B-factors by employing a conjugate gradient minimization using E_{XREF} (Eq. 12.1) as the target. For the isotropic case, the variable is simply the overall B-factor B, and $F_c(\vec{h})$ in Eq. 12.1 is replaced by

$$\exp(\frac{-s^2 B}{4})F_c(\vec{h}) \tag{13.1}$$

For the anisotropic case, the variables are the six parameters of the anisotropic B-factor tensor B_{ij}, and $F_c(\vec{h})$ in Eq. 12.1 is replaced by

$$\exp(-\frac{1}{4}(\sum_{i=1,3} s_{ii}^2 B_{ii} + 2(s_1 s_2 B_{12} + s_1 s_3 B_{13} + s_2 s_3 B_{23})))F_{calc}(\vec{h}) \tag{13.2}$$

Normally, the following restrictions apply to the anisotropic B-factor tensor (see RESTriction statement):

Triclinc	none
Monoclinic	$B_{13} = B_{23} = 0$ when $\beta = \alpha = 90°$
	$B_{12} = B_{23} = 0$ when $\gamma = \alpha = 90°$
	$B_{12} = B_{13} = 0$ when $\gamma = \beta = 90°$
Orthorhombic	$B_{12} = B_{13} = B_{23} = 0$
Tetragonal	$B_{11} = B_{22}$ and $B_{12} = B_{13} = B_{23} = 0$
Rhombohedral	$B_{11} = B_{22} = B_{33}$ and $B_{12} = B_{13} = B_{23}$
Hexagonal	$B_{11} = B_{22}$ and $B_{13} = B_{23} = 0$
Cubic	$B_{11} = B_{22} = B_{33}$ and $B_{12} = B_{13} = B_{23} = 0$.

After termination, the program modifies the F_{calc} structure factors and the B atomic property array (Section 2.16). In the isotropic case, it adds the refined B-factor shift to the B array and it also applies the B-factor shift to the F_{calc} structure factors. In the anisotropic case, the overall isotropic contribution is subtracted from the B_{ij} tensor such that the trace of B is zero. This modified B tensor is then applied to the calculated structure factors. The overall isotropic contribution is added to the B-factor array F_{calc}. Note that F_{part} is not touched during this procedure.

13.2.1 Syntax

OPTImize OVERall { <**xrefin-optimize-overall-statement**> } **END**
is an xrefin statement. Action takes place as soon as the END statement is issued.

<**xrefin-optimize-overall-statement**>:==

ANISotropic=<logical> carries out anisotropic refinement if TRUE; if FALSE, isotropic refinement is carried out (default: FALSE).

B=<real> is a starting overall isotropic B-factor (only for ANISotropic = FALSE) (default: 0).

B11=<real> B22=<real> B33=<real> B12=<real>
B13=<real> B23=<real> is a starting overall anisotropic B-factor (only for ANISotropic=TRUE) (default: 0).

DROP=<real> is the initial expected drop in the target function (default: 10.0).

NSTEp=<integer> is the number of least-squares cycles (default: 5).

TOLErance=<real> specifies the termination criterion: minimal gradient (default: 0).

RESTiction=ALL | OFFDiagonal | NONE specifies if the anisotropic B-factor tensor will be restricted according to the space group symmetry. ALL means that all B parameters are strictly restricted according to the above table. OFFDiagonal means that only B_{12}, B_{13} and B_{23} are restricted. NONE means no restrictions (default: ALL).

13.2.2 Requirements

The diffraction data, calculated structure factors (F_{calc}), unit cell parameters, and selections for the reflections have to be defined. The F_{calc} structure factors must be read or computed prior to invoking this routine, e.g., through the update xrefin statement.

13.2.3 Example: Refinement of Overall Anisotropic B-Factors

Prior to invoking the refinement and after the refinement, the F_{calc} and F_{obs} arrays are swapped. This implies that the B-factor correction is applied to the F_{obs}. The corrected F_{obs} are then written to a new reflection file. To refine an isotropic B-factor, simply set the ANISotropic option to FALSE, and write the coordinates with the modified B-factors to a file instead of writing the scaled F_{obs}.

```
1 remarks file xtalrefine/baoverall.inp
2 remarks Overall anisotropic B-factor refinement
3 remarks This job will produce a scaled data set
4
5 {===>} parameter @TOPPAR:parhcsdx.pro end                {*Read parameters.*}
```

```
6
7  {===>} structure @../generate/generate.psf end       {*Read structure file.*}
8
9  {===>} coor @slowcool.pdb                              {*Read coordinates.*}
10
11
12  xrefine                                      {*This invokes the            *}
13                                               {*crystallographic data parser.*}
14 {===>}
15    a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}
16
17 {===>}
18    symmetry=(x,y,z)                           {*Symmetry operators for space *}
19    symmetry=(-x+1/2,-y,z+1/2)                 {*group P212121; notation as in*}
20    symmetry=(-x,y+1/2,-z+1/2)                 {*Int. Tables.                 *}
21    symmetry=(x+1/2,-y+1/2,-z)
22
23    SCATter ( chemical C* )
24       2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
25    SCATter ( chemical N* )
26       12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
27    SCATter ( chemical O* )
28       3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
29    SCATter ( chemical S* )
30       6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
31    SCATter ( chemical P* )
32       6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
33    SCATter ( chemical FE* )
34       11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070
35
36 {===>}
37    nreflections=15000
38    reflection @amy.fob end                              {*Read reflections.*}
39
40 {===>}
41    resolution  5.0 2.0                                  {*Resolution range.*}
42
43    reduce
44    do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
45    fwind=0.1=100000
46
47    method=FFT
48
49    fft
50       memory=1000000
51    end
52
53
54    tolerance=0.0 lookup=false             {*This makes the minimizer happy.*}
55
56  end
57
58  xrefin
59    update-fcalc                             {*Compute fcalcs with current *}
60                                             {*B-factors stored in array B.*}
61    print rfactor
62
63    do (fpart=fcalc)                           {*Swap Fcalc and Fobs       *}
64    do (fcalc=fobs)                            {*(caution: Fpart is reset).*}
65    do (fobs=fpart)
66    do (fpart=0.)
67
68    optimize overall-bfactor
69       nstep=20                                   {*Do 8 steps of conjugate*}
70       anisotropic=true                           {*gradient minimization. *}
71       b11=0. b22=0. b33=0. b12=0. b13=0.
72       b23=0.
73    end
74                                             {*The routine will apply the*}
75                                             {*overall B-factor shift to *}
```

```
76                                                {*the Fcalcs.              *}
77
78    do (fpart=fcalc)                                  {*Swap FCALC and FOBS.*}
79    do (fcalc=fobs)
80    do (fobs=fpart)
81    do (fpart=0.)
82
83    print rfactor
84
85    write
86      reflections fobs                          {*Write scaled FOBS to file.*}
87
88 {===>}
89      output=fobs_scaled.fob
90    end
91  end
92
93  stop
```

13.3 Grouped B-Factor and Occupancy Refinement

The xrefin optimize group statement optimizes grouped B-factors and occupancies, e.g., two for each residue. The procedure minimizes the function E_{XREF}. The Powell conjugate gradient minimization method is used. B-factors and occupancies can be refined simultaneously by specifying both the B and the Q options.

Uniform shifts are applied to the B-factors and occupancies for each of the selected groups. The optimization is started using the current values of the B-factors and occupancies, regardless of whether these values are identical within a particular group.

13.3.1 Syntax

OPTImize GROUp { <**xrefin-optimize-group-statement**> } **END**
is an xrefin statement. Action takes place as soon as the END statement is issued.

<**xrefin-optimize-group-statement**>:==

B=<selection> adds a selected group of atoms to the database for B-factor refinement (default: none). All atoms in the group assume the same B-factor. After the refinement has been carried out, the selections are erased.

BFMIn=<real> specifies the minimum B-factor allowed (default: 2.0 $Å^2$).

DROP=<real> specifies the initial expected drop in energy (default: 10.0).

NSTEp=<integer> is the number of conjugate gradient steps (default: 0).

Q=<selection> adds a selected group of atoms to the database for occupancy refinement (default: none). All atoms in the group assume the same occupancy. After the refinement has been carried out, the selections are erased. No constraints (e.g, $q_1 + q_2 = 1$) are possible at present.

TOLErance=<real> is the gradient that forces exit (default: 0).

13.3.2 Requirements

The molecular structure, atomic B-factors, diffraction data, atomic-form factors, and unit cell parameters have to be defined.

13.3.3 Example

```
 remarks  file  xtalrefine/bgroup.inp
 remarks  Grouped, unrestrained B-factor refinement

{===>} parameter @TOPPAR:parhcsdx.pro end                {*Read parameters.*}

{===>} structure @../generate/generate.psf end       {*Read structure file.*}

{===>} coor @slowcool.pdb                               {*Read coordinates.*}


 xrefine                                       {*This invokes the             *}
                                               {*crystallographic data parser.*}
{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                            {*Symmetry operators for space *}
   symmetry=(-x+1/2,-y,z+1/2)                  {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)                  {*Int. Tables.                 *}
   symmetry=(x+1/2,-y+1/2,-z)

   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=15000
   reflection @amy.fob end                              {*Read reflections.*}

{===>}
   resolution  5.0 2.0                                  {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000

   method=FFT

   fft
```

```
49        memory=1000000
50      end
51
52
53      tolerance=0.0 lookup=false              {*This makes the minimizer happy.*}
54
55    end
56
57    xrefin
58      optimize group
59
60        {*The following loop selects two B-factor groups for each residue,   *}
61        {*one for backbone (ca, n, c) and one for side-chain atoms.          *}
62        {*The program knows that it has to go through all residues, since the*}
63        {*loop goes over all ca atoms.                                       *}
64        {*In principle, you can select any arbitrary group of B-factors here.*}
65
66        for $1 in id ( name ca ) loop main
67          b=(( name ca or name n or name c ) and byres id $1 )
68          b=( not ( hydro or name ca or name c or name n ) and
69              byres id $1 )
70        end loop main
71
72        nstep=15                                  {*Fifteen steps should be *}
73        drop=1.0                                 {*sufficient in most cases.*}
74        ?
75      end
76    end
77
78
79    write coordinates output=bgroup.pdb end           {*Write coordinates with*}
80                                                      {*B-factors.            *}
81    stop
```

13.4 Individual B-Factor Refinement

This statement optimizes individual (restrained) isotropic B-factors. Distinctions between backbone and side-chain B-factor restraints can be made by appropriate selections. B-factors of atoms are both selected in xrefin and free to move. The procedure minimizes the target function

$$T = E_{XREF} + E_R \tag{13.3}$$

where the restraining term E_R is given by

$$\begin{aligned} E_R \;=\; & W_B \sum_{(i,j)-bonds} (B_i - B_j)^2/{\sigma_{bonds}}^2 \\ & + W_B \sum_{(i,j,k)-angles} (B_i - B_k)^2/{\sigma_{angles}}^2 \\ & + W_B \sum_{k-group} \sum_{j-equivalences} \sum_{i-unique\ atoms} (B_{ijk} - \bar{B}_{ik})^2/{\sigma_{ncs}}^2 \end{aligned} \tag{13.4}$$

The summations are carried out over all bond and angle terms present in the molecular structure that involve selected and free atoms. The last sum is used only if non–crystallographic symmetry restraints are present (see Chapter 16). The Powell conjugate gradient minimization method is used.

The following symbols are declared at the end of the refinement. In order to get these statistical values without refinement set NSTEP to -1.

$BRMS_BOND overall rmsd for bond B restraints

$BN_BOND total number of bond B restraints

$NGROUP_BOND number of groups of bond restraints

$BRMS_BOND_$k rmsd for bond B restraints for group k

$BN_BOND_$k number of bond B restraints for group k

$NGROUP_ANGL number of groups of angle restraints

$BRMS_ANGL overall rmsd for angle B restraints

$BN_ANGL total number of angle restraints

$BRMS_ANGL_$k rmsd for angle B restraints for group k

$BN_ANGL_$k number of angle B restraints for group k

$NGROUP_NCS number of groups of NCS restraints

$BRMS_NCS_$k rmsd for NCS B restraints for group k

$BN_NCS_$k number of NCS B restraints for group k

13.4.1 Syntax

OPTImize BFACtor { <**xrefin-optimize-bfactor-statement**> } **END**
is an xrefin statement. Action takes place as soon as the END statement is issued.

<**xrefin-optimize-bfactor-statement**>:==

ASIGma=<selection>=<real> specifies the target standard deviation between B-factors of selected atoms connected by an angle σ_{angles}. Multiple specification is possible (default: none).

BFMIn=<real> specifies the minimum B-factor allowed (default: 2.0 Å^2).

BSIGma=<selection>=<real> specifies the target standard deviation between B-factors of selected bonded atoms σ_{bonds}. Multiple specification is possible (default: none).

DROP=<real> is the initial expected drop in energy (default: 10.0).

NSTEp=<integer> is the number of conjugate gradient steps (default: 0).

RWEIght=<real> is the weight W_B for B-factor restraints. If negative, weight will be set automatically. If zero, no restraints will be computed (default: -1).

TOLErance=<real> is the gradient that forces exit (default: 0).

13.4.2 Requirements

The molecular structure, atomic B-factors, diffraction data, atomic-form factors, and unit cell parameters have to be defined.

13.4.3 Example

The following example performs overall B-factor refinement, followed by restrained individual B-factor refinement:

```
 remarks  file  xtalrefine/brefinement.inp
 remarks  Restrained, individual B-factor refinement


{===>} parameter @TOPPAR:parhcsdx.pro end                {*Read parameters.*}

{===>} structure @../generate/generate.psf end       {*Read structure file.*}

{===>} coor @slowcool.pdb                               {*Read coordinates.*}


 xrefine                                        {*This invokes the             *}
                                                {*crystallographic data parser.*}
{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                             {*Symmetry operators for space *}
   symmetry=(-x+1/2,-y,z+1/2)                   {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)                   {*Int. Tables.                 *}
   symmetry=(x+1/2,-y+1/2,-z)

   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=15000
   reflection @amy.fob end                              {*Read reflections.*}

{===>}
   resolution  5.0 2.0                                  {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000

   method=FFT

   fft
      memory=1000000
   end


   tolerance=0.0 lookup=false             {*This makes the minimizer happy.*}

   wa=10000
```

```
57
58  end
59
60
61  xrefin
62    update
63    optimize overall                          {*Optimize overall B-factor.*}
64       drop=0.01                      {*This avoids overshooting the minimum.*}
65       nstep=15
66    end
67  end
68
69               {*The following two VECTOR statements make Gaussian          *}
70               {*distributions around current B-factors (stored in array B),*}
71               {*with width 1.5 for backbone and 2.0 for side chains.       *}
72
73  vector do (b=b + gauss(1.5))
74     ( name ca or name n or name c )
75
76  vector do (b=b + gauss(2.0))
77     ( not hydrogen and not ( name ca or name n or name c ))
78
79  xrefin
80                                                {*Used during refinement.*}
81
82    optimize bfactors                          {*Invoke B-factor optimizer.*}
83
84       nstep=20                         {*Twenty steps of conjugate gradient.*}
85
86       drop=10.0                           {*Expected initial drop in energy.*}
87
88         {*By default, the program will automatically determine the relative*}
89         {*weight between restraints and crystallographic residual.         *}
90
91                                       {*Target sigma for 1-2 B-factor pairs*}
92                                       {*(for backbone and side chain).     *}
93       bsigma=( name ca or name n or name c or name o )=1.5
94       bsigma=( not( name ca or name n or name c or name o ))=2.0
95
96                                            {*Target sigma for 1-3 (angle)*}
97                                            {*B-factor pairs.              *}
98       asigma=( name ca or name n or name c or name o )=2.0
99       asigma=( not( name ca or name n or name c or name o ))=2.5
100
101    end                                           {*Refinement is executed.*}
102
103 end
104
105 write coordinates output=brefinement.pdb end     {*Write final coordinates*}
106                                                  {*and B-factors.         *}
107
108 stop
```

The B-factors can be plotted as a function of residue number by first running the following X-PLOR protocol, which produces a text file "bfactorplot.list":

```
1  remarks file xtalrefine/bfactorplot.inp -- B-factors per residue
2  remarks specific for proteins: backbone and side chain atoms.
3
4                                                    {*Read structure file.*}
5   {===>} structure @../generate/generate.psf end
6
7   {===>} coordinates @brefinement.pdb                 {*Read coordinates.*}
8
9   {===>} vector ident (store9) ( tag )      {*This selects all residues to  *}
10                                            {*be analyzed.  Change it       *}
11                                            {*to analyze portions,          *}
```

```
                                          {*e.g., ( tag and segid "A" ).  *}
                                          {*"tag" assigns one unique atom *}
                                          {*per residue (see <selection>).*}


  {===>} set display=bfactorplot.list end   {*Write the plot information to*}
                                            {*the specified file.          *}

  set echo=off end                        {*Turn off echo to reduce output.*}
  set message=off end                          {*Turn off warning messages.*}

  evaluate ($number=0)
  for $atom_id in id ( store9 ) loop out2
      vector show norm ( b ) ( byresidue ( id $atom_id )
                               and ( name ca or name n or name c ) )
      evaluate ( $back=$result )
      vector show norm ( b ) ( byresidue ( id $atom_id )
                               and not ( name ca or name n or name c )
                               and not hydrogen )
      evaluate ( $side=$result )
      vector show elem ( resid ) ( id $atom_id )

      evaluate ($number=$number+1)
      display   $number $back $side
  end loop out2

  set echo=on end
  set message=on end

  stop
```

In the second step, a Mathematica script file can be used to produce the B-factor plots shown in Fig. 13.2 for both backbone and side-chain atoms:

```
        (* file xtalrefine/bfactorplot.math  -- make a plot b vs residue   *)

Off[General::spell1];
$DefaultFont={"Times-Roman",13};

                                      (* Change the name of the input file. *)
 (*===>*)    data=ReadList["bfactorplot.list",{Number,Number,Number}];

     column1 = Transpose[data][[1]];
     column2 = Transpose[data][[2]];
     column3 = Transpose[data][[3]];
     max=Max[column2,column3];
     number=Dimensions[column1][[1]];
     ListPlot[column2,
      Axes->True,
      AxesLabel->{"Seq. Res. No.","B-factor"},
      Ticks->{Range[0, number , 5],Range[0.0, max ,5.]},
      PlotLabel->"backbone", AxesOrigin->{0,0},PlotJoined->True,
      PlotRange->{{0, number +2 }, {0., max +2 }}];
     ListPlot[column3,
      Axes->True,
      AxesLabel->{"Seq. Res. No.","B-factor"},
      Ticks->{Range[0, number , 5],Range[0.0, max ,5.]},
      PlotLabel->"side chain", AxesOrigin->{0,0},PlotJoined->True,
      PlotRange->{{0, number +2 }, {0., max +2 }}];
```

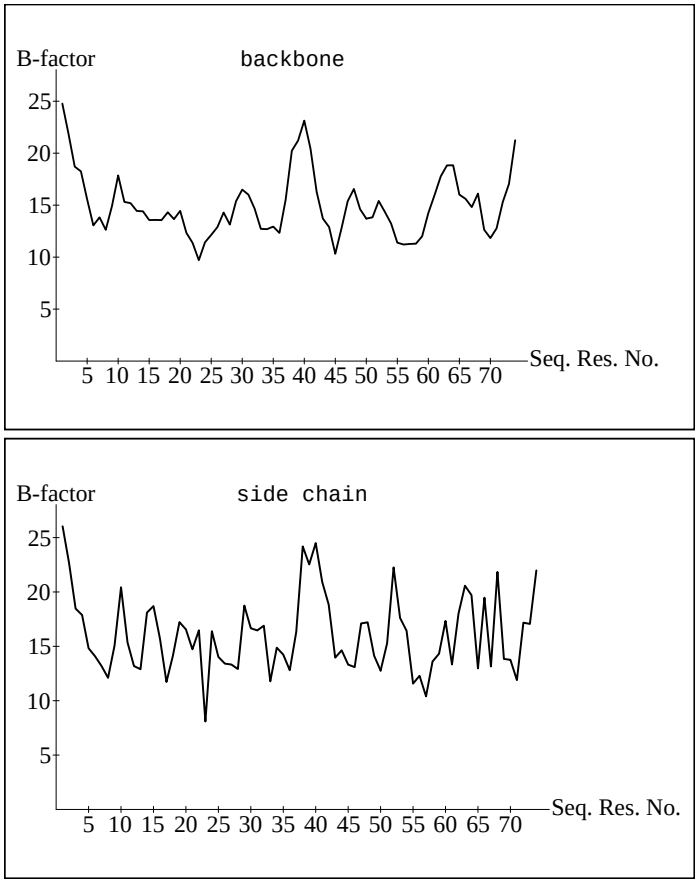

Figure 13.2: B-factor vs. residue number.

13.5 Analysis of Refined Structures

Refer to Chapter 5 for an analysis of the geometric and energetic properties of the coordinates (e.g., Ramachandran plots) and to Section 12.12 for an analysis of the R value distribution.

14 Electron Density Maps

The xrefin map statement computes an electron density map from $F_c = F_{calc} + F_{part}$. F_c structure factors belonging to the test set are set to zero (see Chapter 15). An FFT method is used at the specified resolution. The user can specify boundaries to reduce the size of the map that is written to the output file. Of particular interest is the MOLEcule=EXTEnd option to write a map that covers selected atoms of the molecule.

14.1 Syntax

MAP { <**xrefin-map-statement**> } **END** is an xrefin statement. Action takes place as soon as the END statement is issued.

<**xrefin-map-statement**>:==

AUTOmatic=<logical> is a flag indicating that automatic scaling is performed. The electron density map will have an average of zero and a standard deviation (σ) of one (default: TRUE).

CUSHion=<real> specifies additional cushioning around the molecule in Å. Only for EXTEnd=MOLEcule mode (default: 2.0 Å).

EXTEnd=MOLEcule | UNIT | ORTHogonal_box | FRACtional_box is a mode for boundary specification (default: MOLEcule). In MOLEcule mode, the map covers the selected atoms of the molecule. In UNIT mode, the whole unit cell is written. In ORTHogonal_box mode, the user can specify the boundaries of a cubic box in orthogonal Å. In FRACtional_box mode, one specifies the boundaries in fractional coordinates.

FORMatted=<logical> is a flag that indicates whether a formatted (FORM=TRUE) or binary (FORM=FALSE) file is written (default: TRUE).

IOUTput=<filename> is a filename for the imaginary portion of the electron density map; it works only when HERMitian is set to FALSE (cf. Section 12.3) (default: no output).

OUTPut=<filename> is the filename for the real portion of the electron density map (default: standard output).

SELEction=<selection> selects atoms for EXTEnd=MOLEcule mode. This selection is independent from the selection in the xrefin statement. The map is written to cover the selected atoms (default: all).

XMAX is only for EXTEnd=ORTHogonal or EXTEnd=FRACtional mode; it specifies upper x boundary of box in orthogonal Å or fractional coordinates (default: 1.0).

XMIN is only for EXTEnd=ORTHogonal or EXTEnd=FRACtional mode; it specifies lower x boundary of box in orthogonal Å or fractional coordinates (default: 0.0).

YMAX is only for EXTEnd=ORTHogonal or EXTEnd=FRACtional mode; it specifies upper y boundary of box in orthogonal Å or fractional coordinates (default: 1.0).

YMIN is only for EXTEnd=ORTHogonal or EXTEnd=FRACtional mode; it specifies lower y boundary of box in orthogonal Å or fractional coordinates (default: 0.0).

ZMAX is only for EXTEnd=ORTHogonal or EXTEnd=FRACtional mode; it specifies upper z boundary of box in orthogonal Å or fractional coordinates (default: 1.0).

ZMIN is only for EXTEnd=ORTHogonal or EXTEnd=FRACtional mode; it specifies lower z boundary of box in orthogonal Å or fractional coordinates F(default: 0.0).

14.2 Requirements

The structure factors (F_{obs} and $F_c = F_{calc} + F_{part}$), atomic-form factors, selections for reflections, and unit cell parameters have to be defined.

14.3 Electron Density Map File

The output file of the xrefin map statement contains information about the unit cell, indexing of the map, and the actual electron density map. It can be read by a modified version of MAPPAGE to provide a "DSN6" file for FRODO. Modified versions of this program are included in the directory "[.VAX.FRODOMAP]" for the VAX/VMS system and "sgi/frodomap" for the SGI systems. The grid size of the map is determined by the GRID parameter in the FFT statement. The following is a FORTRAN example of how to read the formatted (QFORM=.TRUE.) or unformatted (QFORM=.FALSE.) version of the electron density maps:

```
1 C
2 C read title
3       IF (QFORM) THEN
4       READ(UNIT,'(/I8)',END=6,ERR=7) NTITLE
5       IF (NTITLE .LE. 0) THEN
```

```
      READ( UNIT, '(A)',END=6,ERR=7)
      ELSE
      DO J = 1, NTITLE
      TITLE(J) = ' '
      READ (UNIT, '(A)',END=6,ERR=7) TITLE(J)
      ENDDO
      ENDIF
      ELSE
      DO J=1,MXTITL
      TITLE(J)=' '
      END DO
      READ(UNIT,END=6,ERR=7) NTITLE,(TITLE(J)(1:80),J=1,NTITLE)
      END IF
C
C read sectioning information
      IF (QFORM) THEN
      READ(U,'(9I8)',END=6,ERR=7)
     &  NA,AMIN,AMAX,NB,BMIN,BMAX,NC,CMIN,CMAX
      ELSE
      READ(U,END=6,ERR=7)
     &  NA,AMIN,AMAX,NB,BMIN,BMAX,NC,CMIN,CMAX
      END IF
C
C read unit cell constants in angstroms and degrees
      IF (QFORM) THEN
      READ(U,'(6E12.5)',END=6,ERR=7) (CELL(I),I=1,6)
      ELSE
      READ(U,END=6,ERR=7) (CELL(I),I=1,6)
      END IF
      END IF
C
C read matrix mode
      IF (QFORM) THEN
      READ(U,'(3A)',END=6,ERR=7) MODE
      ELSE
      READ(U,END=6,ERR=7) MODE
      END IF
      IF (MODE.NE.'ZYX') THEN
      CALL WRNDIE(-5,'RMAP','error in matrix mode')
      GOTO 7
      END IF
C
C read density matrix, c is slowest ("z-sections").
      DO C=CMIN,CMAX
C
C read next section
      IF (QFORM) THEN
      READ(U,'(I8)',END=6,ERR=7) KSECT
      READ(U,'(6E12.5)',END=6,ERR=7)
     &  ((MAP(A,B,C),A=AMIN,AMAX),B=BMIN,BMAX)
      ELSE
      READ(U,END=6,ERR=7) KSECT
      READ(U,END=6,ERR=7)
     &  ((MAP(A,B,C),A=AMIN,AMAX),B=BMIN,BMAX)
      END IF
      END DO
```

The following lines show the beginning of a typical electron density map file:

```

       2 !NTITLE
 REMARKS FILENAME=""
 REMARKS DATE:18-Jun-90  12:24:08       created by user:
      30       4      12      15       5      10      16       2      12
 0.40960E+02 0.18650E+02 0.22520E+02 0.90000E+02 0.90770E+02 0.90000E+02
ZYX
       0
```

```
-0.97086E+00-0.49927E+00-0.82774E+00-0.13491E+01-0.57034E+00-0.71253E-01
-0.19491E+00 0.61017E+00 0.10064E+01-0.22888E+01-0.94020E+00 0.77451E+00
 0.57539E+00-0.31211E-01-0.27430E+00-0.36526E+00 0.34772E+00 0.81884E+00
-0.19954E+01-0.10117E+01 0.18038E+01 0.19008E+01 0.11886E+00-0.41646E+00
 0.47560E-01 0.48855E+00 0.57606E+00-0.22320E+00-0.12787E+01 0.47590E+00
```

14.4 Example: Computation of a $2F_{obs} - F_{calc}$ Map

```
 remarks  file  xtal_maps/map.inp
 remarks  Compute electron density maps of the type  ( a Fobs - b Fobs )

{===>} structure @../generate/generate.psf end       {*Read structure file.*}

{===>} coor @../xtalrefine/slowcool.pdb                {*Read coordinates.*}

 xrefine

{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                          {*Symmetry operators for space *}
   symmetry=(-x+1/2,-y,z+1/2)                {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)                {*Int. Tables.                 *}
   symmetry=(x+1/2,-y+1/2,-z)

   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=15000
   reflection @../xtalrefine/amy.fob end               {*Read reflections.*}

{===>}
   resolution  10. 2.0                                {*Resolution range. *}
                                    {*We use a lower limit than was used in *}
                                    {*refinement to improve connectivity.   *}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000

   method=FFT

   fft
      memory=1000000
   end


 end

xrefin
   update                                                {*Compute FCALCS.*}

   do complex (FCALC=FCALC+FPART)         {* Add FPART to FCALC explicitly *}
```

```
58    do complex (FPART=0)                    {* to allow proper scaling.      *}
59
60    do scale ( FCALC = FOBS )                     {*FCALCS is scaled to FOBS.*}
61
62    do amplitude (FCALC = 2*FOBS - FCALC)         {*This is for a 2Fo-Fc map.*}
63                                                  {*Any other combination of *}
64                                                  {*coefficients can be used.*}
65
66    map                                                 {*Invoke map parser. *}
67 {===>}
68       extend=molecule                         {*Computed map covers the        *}
69       cushion=2.0                             {*selected atoms of the molecules*}
70       selection=( resid 1:40 )                {*in the asymmetric unit with a  *}
71                                               {*2.0 A cushion around it.       *}
72
73 {===>}  output=map.2fofc                                      {*Output file.*}
74
75     end                                           {*Map is being computed and*}
76                                                  {*written to file.         *}
77  end
78
79  stop
```

14.5 Example: Computation of an Omit Map

The following example shows how to compute a "ordinary" omit map, where residue 20 has been left out of the calculation:

```
1  remarks  file  xtal_maps/omit_map.inp
2  remarks  Compute omit maps of the type  ( a Fobs - b Fobs )
3
4 {===>} structure @../generate/generate.psf end       {*Read structure file.*}
5
6 {===>} coor @../xtalrefine/slowcool.pdb                 {*Read coordinates.*}
7
8  xrefine
9
10 {===>}
11    a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}
12
13 {===>}
14    symmetry=(x,y,z)                          {*Symmetry operators for space *}
15    symmetry=(-x+1/2,-y,z+1/2)                {*group P212121; notation as in*}
16    symmetry=(-x,y+1/2,-z+1/2)                {*Int. Tables.                 *}
17    symmetry=(x+1/2,-y+1/2,-z)
18
19    SCATter ( chemical C* )
20       2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
21    SCATter ( chemical N* )
22       12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
23    SCATter ( chemical O* )
24       3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
25    SCATter ( chemical S* )
26       6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
27    SCATter ( chemical P* )
28       6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
29    SCATter ( chemical FE* )
30       11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070
31
32 {===>}
33    nreflections=15000
34    reflection @../xtalrefine/amy.fob end                {*Read reflections.*}
35
36 {===>}
37    resolution  10. 2.0                                  {*Resolution range. *}
38                                    {*We use a lower limit than was used in *}
39                                    {*refinement to improve connectivity.   *}
```

```
40
41    reduce
42    do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
43    fwind=0.1=100000
44
45    method=FFT
46
47    fft
48       memory=1000000
49    end
50
51
52  end
53
54 xrefin
55
56
57 {===>}  selection=( not resid 20 )           {*Select atoms to be included;*}
58                                              {*in this case, residue 20 is *}
59                                              {*omitted.                    *}
60    update
61
62    do complex (FCALC=FCALC+FPART)         {* Add FPART to FCALC explicitly *}
63    do complex (FPART=0)                   {* to allow proper scaling.      *}
64
65    do scale ( FCALC = FOBS )
66    do amplitude (FCALC = 2*FOBS - FCALC) {*Other coefficients are possible.*}
67
68
69
70    map
71       extend=molecule
72       cushion=6.0
73 {===>}  selection=( resid 20 )        {*Make map that covers residue 20 and*}
74                                       {*surroundings.                       *}
75
76
77 {===>}  output=omit_map.2fofc                                  {*Output file.*}
78
79     end
80  end
81
82  stop
```

14.6 Example: Computation of an Annealed Omit Map

The following example shows how to compute an SA-refined omit map (Hodel, Kim, and Brünger 1992). The omitted region consists of an 8 Å sphere around residue 20. Other selections can be specified here. The immediately surrounding atoms, consisting of a 3 Å shell, are restrained to avoid artificial movement into the omitted region. The slow-cooling protocol is used for simulated annealing (see Section 13.1.3).

```
1 remarks  file  sa_omit_map.inp
2 remarks  Make an annealed, sigmaa-weighted 2fo-fc omit map
3
4
5 {===>}                                                    {* Resolution limits. *}
6 eval ($low_res=6.)                                          {* For refinement. *}
7 eval ($low_res_map=50.)                                  {* For map calculation. *}
8 eval ($high_res=2.3)
```

```

{===>} parameter @TOPPAR:parhcsdx.pro end                              {*Read parameters.*}

{===>} structure @../../tutorial/generate/generate.psf end      {* Read structure file.*}

{===>} coor @../../tutorial/xtalrefine/slowcool.pdb               {* Read coordinates.*}

{===>}
 evaluate ($init_temp=1000. )                                      {*Starting temperature.*}

{===>}                                                             {* Minimization steps. *}
eval ($nstep_mini=20)  !20


{===>}
 vector do (charge=0) ( all )                                   {* Turn off all charges. *}


 flags
    include pele pvdw xref
    ?
 end

 xrefine

{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0              {* Unit cell.*}

{===>}
   eval ($SG="P2(1)2(1)2(1)")

   @SYMMETRY:spacegroup.lib

   @SYMMETRY:scatter.lib

{===>}
   nreflections=15000
   reflection @../../tutorial/xtalrefine/amy.fob end              {* Read reflections.*}


  resolution $low_res $high_res

{===>}
   do ( fobs=0)  ( ampl(fobs) <= 2.0*sigma)                {*Sigma cutoff.*}
   fwind=0.1=100000

   method=FFT

   fft
      memory=1000000
   end


   tolerance=0.2                                    {*Tolerance for dynamics.*}

{===>}
   wa=26000                                      {*Weight from job "check.inp".*}
 end


{===>}                                                {*Define omitted region.*}
vector ident ( store1 ) ( byresidue ( residue 20 around 8. ) )

                         {*Define immediate surroundings of omitted region.*}
vector ident ( store2 )  ( byres ( ( store1 ) around 3.0 and not store1 ) )

                       {*Exclude omitted region for empirical energy terms.*}
constraints interaction=( not store1  )=( not store1 ) end

              {*Keep omitted region fixed during dynamics and minimization.*}
```

```
constraints fix=( store1 ) end

                                    {*Exclude omitted region for X-ray term.*}
xrefin
   selection=( not store1 and not hydrogen)
end


                      {*Apply harmonic restraints to immediate surroundings.*}
vector do ( harmonic=10. ) ( store2 )
vector do ( refx=x ) ( all )
vector do ( refy=y ) ( all )
vector do ( refz=z ) ( all )
flags include harm end


          {*The following dynamics and minimization sections represent the*}
          {*slow-cooling SA protocol.                                      *}

 set seed=432324368 end {*Set the initial random seed for the v-assignment.*}

 vector do (vx=maxwell($init_temp)) ( all )
 vector do (vy=maxwell($init_temp)) ( all )
 vector do (vz=maxwell($init_temp)) ( all )

 vector do (fbeta=100.) ( all )

 evaluate ($1=$init_temp)
 while ($1 > 300.0) loop main

    dynamics  verlet
      timestep=0.0005
      nstep=50
      iasvel=current
      nprint=5  iprfrq=0
      tcoupling=true tbath=$1
    end
    evaluate ($1=$1-25)
 end loop main

 xrefin
   tolerance=0.0    lookup=false          {*This makes the minimizer happy.*}
 end

 minimize powell                                        {*Final minimization.*}
   nstep=$nstep_mini
   drop=10.0
 end

                                   {*Compute the 2 fobs - fcalc SA omit map.*}
 xrefin

     resolution  $low_res_map 2.0                       {*Resolution range. *}
                                    {*We use a lower limit than was used in *}
                                    {*refinement to improve connectivity.   *}
     update



{===>}                                                                {* Scaling. *}
   do (fcalc=scale[bins=1](fobs,fcalc)*fcalc)
                  ($low_res >= d >= $high_res and ampl(fobs) > 0 )

                                                                  {* Sigmaa weighting. *}

   do (OA=norm(amplitude(FOBS)))    ( $low_res >= d >= $high_res and ampl(fobs) > 0 )
   do (OB=norm(amplitude(FCALC)))   ( $low_res >= d >= $high_res and ampl(fobs) > 0 )
   do (OC=SIGA(OA,OB))              ( $low_res >= d >= $high_res and ampl(fobs) > 0 )
   do (OD= OC sqrt( save(amplitude(fobs)^2)  / save(amplitude(fcalc)^2) ) )
```

```
                                    ( $low_res >= d >= $high_res and ampl(fobs) > 0 )
   do (fom= 2 OC OA OB / ( max(0.0001,1-OC^2) ))
                                    ( $low_res >= d >= $high_res and ampl(fobs) > 0 )
   do (fom= i1overi0(fom))                              ( acentric and ampl(fobs) > 0 )
   do (fom= tanh(fom/2))                                ( centric  and ampl(fobs) > 0 )


                                                {* Compute FT of (fom 2fo- D fc) map. *}
   do (deri1 = combine(2 fom ampl(fobs) - OD ampl(fcalc), phase(fcalc)) )
              ( $low_res >= d >= $high_res and acentric and ampl(fobs) > 0 )
   do (deri1 = combine(fom ampl(fobs), phase(fcalc)) )
              ( $low_res >= d >= $high_res and centric and ampl(fobs) > 0 )

   do (map1=ft(deri1)) ( $low_res >= d >= $high_res and ampl(fobs) > 0 )


    remarks  sigmaa-weighted 2fofc map

    write map

      from=map1

{===>}                                      {* Selection of map region to be written. *}
      extend=molecule
      cushion=6.0
      selection=( store1 )


{===>}  output=sa_omit_map.map                                        {*Output file.*}

    end

 stop
```

14.7 Example: Difference Map

The following example shows how to compute a difference map between the native data set ("amy.fob") and an isomorphous data set ("derivative.fob"), e.g., a derivative data set. Phases are obtained from the model. Alternatively, phases can be read as part of the the native data set; in this case the UPDAte statement has to be removed. Note that the step function computes the differences only for the common subset of the native and the other data set.

```
remarks file xtal_maps/diff_map.inp   --

      {* Read structure file and coordinates if phases should be computed *}
      {* from a model.  Otherwise, phases have to be read from file       *}
      {* "native.fob" (see below).                                        *}

{===>} structure @../generate/generate.psf end       {*Read structure file.*}

{===>} coor @../xtalrefine/slowcool.pdb               {*Read coordinates. *}

 xrefine

{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                         {*Symmetry operators for space *}
```

```
   symmetry=(-x+1/2,-y,z+1/2)              {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)              {*Int. Tables.                 *}
   symmetry=(x+1/2,-y+1/2,-z)
{===>}
   asymm=( 0<=x<=1/2                         {* Asymmetric unit for maps. *}
      and  0<=y<=1/2
      and  0<=z<=1 )



   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070
   SCATter ( chemical ZN* )
      11.9719 2.99460 7.38620 0.20310 6.46680 7.08260 1.39400 18.0995
      0.78070

{===>}
   nreflections=15000       {* Allocate sufficient space for reflections. *}


{===>}
                                  {* Read native data set, stored in FOBS. *}
   reflection @../xtalrefine/amy.fob end
   do (fpart=fobs)    (all)                  {*Store native data in FPART.*}


{===>} {* If a model structure is present, phases can be computed from the *}
       {* structure and stored in Fcalc.  If other phases other should be  *}
       {* used instead remove the following two lines and read the phases  *}
       {* as FCALc using the reflection statement.                         *}
   resolution  10. 3.0                     {*Resolution range for FCALC. *}
   update

{===>}
                         {* Read data set of other crystal, stored in FOBS.*}
   do (fobs=0.)
   reflection
        @derivative.fob
   end

                        {* Compute $k2 scale factor (FPART scaled to FOBS). *}

   scale
      set1=fobs  k1=-1  b1=0
      set2=fpart        b2=0
      selection=(ampl(fobs)>0 and ampl(fpart)>0 and ampl(fcalc)>0 and test=0
{===>}                                    {* Resolution range for scaling. *}
                and 10 >= d >= 3.)
   end


                                              {* Compute difference map. *}
   do (map1=ft(  combine( ampl(fobs)-$k2 * ampl(fpart) , phase(fcalc))  ))
               (ampl(fobs)>0 and ampl(fpart)>0 and ampl(fcalc)>0 and test=0
{===>}                                       {* Resolution range for map. *}
                and 10 >= d >= 3.)


                                                    {* Write map to file. *}
   write map
```

```
88        from=map1
89
90  {===>}            {* If no model is present, the map extension needs to be *}
91                    {* explicitly specified using xmin, xmax, ...            *}
92        extend=molecule
93        cushion=2.0
94        selection=( all )
95
96
97  {===>}  output=diff_map.map                                  {*Output file.*}
98
99      end                                          {*Map is being computed and*}
100                                                  {*written to file.         *}
101
102
103 end
104
105 stop
```

15 Cross-validation: The Free R Value

X-PLOR provides the possibility of cross-validation in reciprocal space, as described by Brünger (1992, 1993).

The most common measure of the quality of a crystal structure is the R value (Eq. 12.2). R is closely related to the crystallographic residual (cf. Eq. 12.1)

$$R' = \sum_{h,k,l} (|F_{obs}(h,k,l)| - k|F_{calc}(h,k,l)|)^2 \tag{15.1}$$

which is a linear function of the negative logarithm of the likelihood of the atomic model, assuming that all observations are independent and normally distributed. R can be made arbitrarily small by increasing the number of model parameters and subsequent refinement against R'; i.e., the diffraction data can be overfit without changing the information content of the atomic model.

Crystallographic diffraction data are redundant to some degree; e.g., a small portion of the data can be omitted without seriously affecting the result. Following the statistical concept of cross-validation, the observed reflections are partitioned into a test set T and a working set A (Brünger 1992); that is, T and A are disjoint, and their conjunction is the full set of observed reflections. The value

$$R_T^{free} = \frac{\sum_{(h,k,l)\in T} ||F_{obs}(h,k,l)| - k|F_{calc}(h,k,l)||}{\sum_{(h,k,l)\in T} |F_{obs}(h,k,l)|} \tag{15.2}$$

is referred to as the free R value computed for the T set of reflections. T is omitted in the modeling process; e.g., in the case of crystallographic refinement, the residual to be minimized is given by

$$R'_A = \sum_{(h,k,l)\in A} (|F_{obs}(h,k,l)| - k|F_{calc}(h,k,l)|)^2 \tag{15.3}$$

One would expect R_T^{free} to be less prone to overfitting than R. This concept can be applied to the other statistical quantities available in X-PLOR, such as the standard linear correlation coefficient (Eq. 12.1). It can even be

applied to crystal structures that have already been refined with all diffraction data included: refinement by simulated annealing with T omitted will remove some of the memory toward T.

Both R_T^{free} and the rms difference between the model refined against the complete data set and the model refined against A increase more or less monotonically as a function of the percentage of omitted data. This is to be expected of terms that monitor the validity of a model. R decreases, which is a paradoxical and misleading behavior for an indicator of the model's accuracy. As a compromise between avoiding fluctuations of R_T^{free} and maintaining small rms differences between refined models, obtain T from a random selection of 10% of the observed reflections.

The free R value (or correlation coefficient) is printed along with the conventional R value (correlation coefficient) during all refinement procedures in X-PLOR, including PC-refinement for molecular replacement. In addition, the data analysis can be carried out for both the test set T and the working set A when one is using the "PRINt R", "PRINt PHASe", and "PRINt COMPleteness" statements. The R values or correlation coefficients are stored in the symbols $R, $TEST_R, $CORR, and $TEST_CORR whenever a computation of E_{XREF} has been carried out, e.g, when a "PRINt TARGet" statement has been issued or an energy calculation has been carried out.

The following two example files show how to use the free R value concept in X-PLOR. Basically, none of the example files described in the previous section have to be changed. The only requirement is to create a special reflection file that tells X-PLOR which reflections belong to the test set and the working set. This is indicated by the TEST array. The example file below randomly selects 10% of the data and sets the TEST array to 1 for them. Subsequently, a new reflection file "amy.cv" is written that should be used for all subsequent X-PLOR runs. X-PLOR automatically partitions the data into the working set and the test set whenever the TEST array contains nonzero elements. The reflections with TEST=1 are used for the free R value (correlation) computation.

```
 remarks  file  xtal_free_r/setup_free_r.inp
 remarks  Set up reflection file for free R test

{===>} structure @../generate/generate.psf end       {*Read structure file.*}

{===>} evaluate ($percentage=10.)   {*Define percentage of reflections used*}
                                    {*for test set (10% in this case).     *}
 xrefine

{===>}
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                          {*Symmetry operators for space *}
   symmetry=(-x+1/2,-y,z+1/2)                {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)                {*Int. Tables.                 *}
   symmetry=(x+1/2,-y+1/2,-z)

```

```
   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=15000
   reflection @../xtalrefine/amy.fob end                {*Read reflections.*}

{===>}
   resolution  5.0 2.0                                  {*Resolution range.*}

   do (test= int(random()+$percentage/100.) )

{===>}                          {*Write reflections, including test array.*}
   write reflection fobs sigma test weight fom output=amy.cv end

   print completeness                       {*Print completeness statistics.*}
   ?

end

stop

```

The example file below is a combination of the slow-cooling simulated annealing refinement cycle described in Section 13.1.3 and the restrained B-factor refinement described in Section 13.4. Note that no change was required in the input files except for using the "amy.cv" reflection file.

```
 remarks  file  xtal_free_r/full_refinement.inp
 remarks  Example of free R test: full refinement, consisting
 remarks  of initial minimization, slow-cooling SA, and restrained
 remarks  B-factor refinement

{===>} parameter @TOPPAR:parhcsdx.pro end               {*Read parameters.*}

{===>} structure @../generate/generate.psf end       {*Read structure file.*}

{===>} coor @../generate/generate.pdb                   {*Read coordinates.*}

 vector do ( charge=0.0 ) ( resname LYS and
    ( name ce or name nz or name hz* ) )         {*Turn off charges on LYS.*}
 vector do ( charge=0.0 ) ( resname GLU and
    ( name cg or name cd or name oe* ) )         {*Turn off charges on GLU.*}
 vector do ( charge=0.0 ) ( resname ASP and
    ( name cb or name cg or name od* ) )         {*Turn off charges on ASP.*}
 vector do ( charge=0.0 ) ( resname ARG and
    ( name cd or name *E or name cz or name NH* or name HH* ) )
                                                 {*Turn off charges on ARG.*}

 flags
    include pele pvdw xref
    ?
 end

 xrefine

{===>}
```

```
   a=61.76 b=40.73 c=26.74 alpha=90.0 beta=90.0 gamma=90.0     {*Unit cell.*}

{===>}
   symmetry=(x,y,z)                          {*Symmetry operators for space *}
   symmetry=(-x+1/2,-y,z+1/2)                {*group P212121; notation as in*}
   symmetry=(-x,y+1/2,-z+1/2)                {*Int. Tables.                 *}
   symmetry=(x+1/2,-y+1/2,-z)

   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=15000
   reflection @amy.cv end         {*Read reflections, including test array.*}

{===>}
   resolution  5.0 2.0                                  {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000

   method=FFT

   fft
      memory=1000000
   end


   tolerance=0.2                                 {*Tolerance for dynamics.*}

{===>}
   wa=26000                                  {*Weight from job "check.inp".*}
 end

 minimize powell                                    {*Initial minimization.*}
   nstep=40
   drop=40.0
 end

 set seed=432324368 end {*Set the initial random seed for the v-assignment.*}

{===>}
 evaluate ($init_temp=3000. )                       {*Starting temperature.*}
 vector do (vx=maxwell($init_temp)) ( all )
 vector do (vy=maxwell($init_temp)) ( all )
 vector do (vz=maxwell($init_temp)) ( all )

 vector do (fbeta=100.) ( all )

 evaluate ($1=$init_temp)
 while ($1 > 300.0) loop main

    dynamics  verlet
      timestep=0.0005
      nstep=50
      iasvel=current
      nprint=5  iprfrq=0
      tcoupling=true tbath=$1
    end
```

```
100     evaluate ($1=$1-25)
101  end loop main
102
103
104  xrefin
105    tolerance=0.0    lookup=false          {*This makes the minimizer happy.*}
106  end
107
108  minimize powell                                        {*More minimization.*}
109    nstep=120
110    drop=10.0
111  end
112
113  xrefin
114    update
115    optimize overall                            {*Optimize overall B-factor.*}
116       drop=0.01                       {*This avoids overshooting the minimum.*}
117       nstep=15
118    end
119  end
120
121               {*The following two VECTOR statements make Gaussian         *}
122               {*distributions around current B-factors (stored in array B)*}
123               {*with width 1.5 for backbone and 2.0 for side chains.      *}
124
125  vector do (b=b + gauss(1.5))
126     ( name ca or name n or name c )
127
128  vector do (b=b + gauss(2.0))
129     ( not hydrogen and not ( name ca or name n or name c ))
130
131  xrefin
132                                                  {*Used during refinement.*}
133
134    optimize bfactors                           {*Invoke B-factor optimizer.*}
135
136       nstep=20                           {*Twenty steps of conjugate gradient.*}
137
138       drop=10.0                             {*Expected initial drop in energy.*}
139
140         {*By default, the program will automatically determine the relative*}
141         {*weight between restraints and crystallographic residual.         *}
142
143                                       {*Target sigma for 1-2 B-factor pairs *}
144                                       {*(for backbone and side chain).      *}
145       bsigma=( name ca or name n or name c or name o )=1.5
146       bsigma=( not( name ca or name n or name c or name o ))=2.0
147
148                                             {*Target sigma for 1-3 (angle) *}
149                                             {*B-factor pairs.              *}
150       asigma=( name ca or name n or name c or name o )=2.0
151       asigma=( not( name ca or name n or name c or name o ))=2.5
152
153    end                                        {*Refinement will be executed.*}
154
155
156    update
157    print r                                       {*Print R value statistics.*}
158
159  end
160
161  constraints inter=( not hydrogen ) ( not hydrogen ) end
162
163  print threshold=0.06 bonds           {*Do some simple geometric statistics.*}
164  print threshold=10.0 angles
165  print threshold=20.0 impropers
166
167  write coordinates output=full_refinement.pdb end {*Write final coordinates*}
168                                                   {*and B-factors.         *}
169
```

```
170 stop
```

As a consequence of the SA-refinement with the test set omitted, the free R value deviates from the conventional R value. However, the free R value decreases during the course of the refinement, even though the test set of reflections has been omitted from the refinement process. This indicates that the information content and phase accuracy of the model increase during the refinement process. If at any stage in the refinement process—e.g., after refining additional water molecules—the free R value increased, it would indicate that the phase accuracy of the model was worsened by the additional refinement. The free R value can thus be used to prevent the user from overfitting the diffraction data.

Figure 15.1 was produced by obtaining the free and conventional R values using the UNIX grep facility from the X-PLOR output file (searching for "TEST=1" and "TEST=0"). The resulting lines were fed into a spreadsheet program.

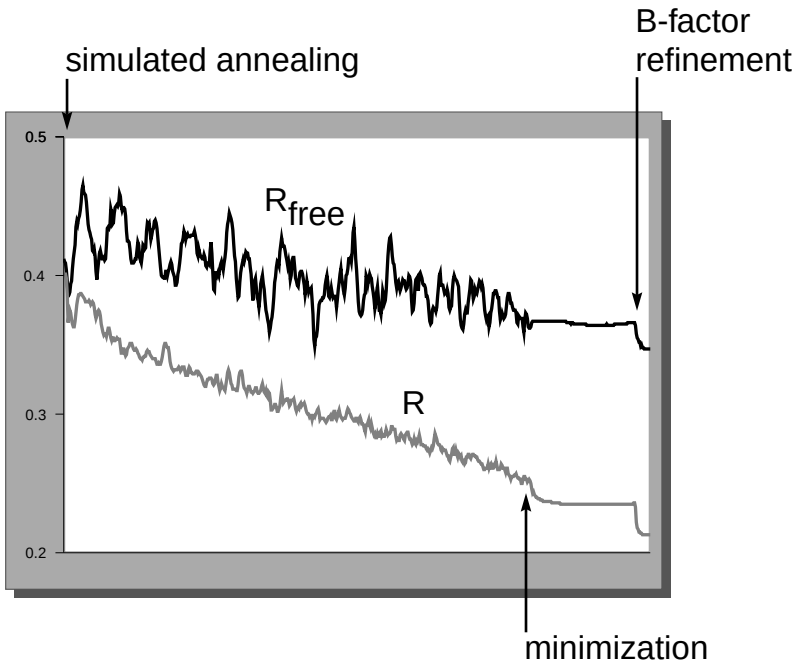

Figure 15.1: Course of refinement.

16 Non-crystallographic Symmetry

Non-crystallographic symmetry (NCS) has been introduced into X-PLOR in two different ways (Weis *et al.*, 1990). In the first, NCS-related atoms are restrained in their average positions by means of an effective energy term analogous to the covalent bond energy. This is the method used, for example, by the refinement program PROLSQ (Hendrickson 1985). The user specifies groups of NCS-equivalent atoms. A least-squares superposition of these atoms onto a reference set (taken as the first set defined) is computed, and the average x,y,z for each atom is taken. Each atom is then restrained according to the energy

$$E_{NCS} = w(x - \bar{x})^2 \tag{16.1}$$

with w used to weight the restraint. The energy flag NCS (see Section 4.5) is used to turn this effective energy term on or off. NCS-related B-factors are similarly restrained to their average value (cf. Section 13.4):

$$(b - \bar{b})^2/{\sigma_{ncs}}^2 \tag{16.2}$$

At present, NCS restraints can be applied only to refinement of individual B-factors; this option is not available for grouped B-factor refinement. Specifications of symmetry-related atoms and the weight factors are given with the NCS restraints statement.

The second way assumes that all NCS-related molecules are strictly identical . Specification of the coordinate transformations defining the NCS is given by the NCS strict statement. All energy calculations, such as minimization and dynamics, are performed on the unique (protomer) atoms only. For the X-ray terms, the NCS coordinate transformations are applied to the input protomer to generate F_{calc}s and derivatives for the entire crystallographic asymmetric unit, and the derivatives are transformed back to the protomer coordinates to apply an NCS-averaged force.

For the case of strict NCS, the nonbonded interactions among NCS-related protomers can be treated in a manner analogous to crystal symmetry contacts: the operators defining the NCS relations are used to compute the nonbonded interactions between the protomer being refined and its neighbors. By the assumptions of strict symmetry, the crystal lattice contacts are

ignored in this option. For structures in which some of the internal symmetry is non-crystallographic and some is crystallographic (e.g., icosahedral viruses), additional operators besides those needed for structure factor calculations (i.e., a subset of the crystallographic operators) can be input to complete the specification of the nonbonded interactions arising from the internal symmetry. The interprotomer nonbonded interactions are turned on by setting the packing flags PVDW and PELE (Section 4.5; these energies are used for the internal symmetry interactions only, as the crystal contacts are automatically shut off).

The NCS strict statement and the NCS restraints statement are independent; i.e., the two options can be used together. In particular, NCS restraints within a protomer can be refined under the STRIct option.

There is a third way to impose symmetry restraints on a dimer based on distance differences (Section 18.8). This alternative allows one to impose exact twofold symmetry on a dimer without specifying the actual operation between the monomers; i.e., the separation between the monomers will be a self-adjusting parameter.

16.1 NCS Restraints

A "group" refers to a set of equivalent atoms. The GROUp statement tells the program that a new set is being input. The EQUIvalence selection serially defines the NCS-related atoms within the group. That is, the first atoms of each equivalence set are related by NCS, the second atoms of each set are related by NCS, etc.; the program superimposes the selected atoms by their position in the equivalence set. X-PLOR demands that each equivalence selection within a group have the same number of atoms, residue names, and atom names; thus NCS-related atoms can be distinguished by segment name or residue number (typically, separate chains are given different segment names).

For example, if a non-crystallographic trimer were being refined with all monomers at the same weight, each monomer would be an equivalence set within a group. In this case, we would have one GROUp statement and three EQUIvalence selections, one for each monomer. If we wanted to restrain the side-chain atoms more loosely than the main chain, two GROUps would be given: the first would have 3 equivalence sets, each selecting a different monomer and the main chain atoms, with one set of weights; the second would also have three equivalence sets, each selecting a different monomer and the side-chain atoms, with a second set of weights. Unrestrained atoms are not selected in the EQUIvalence statement (e.g., those that are known to deviate due to lattice contacts).

A given atom can be used in more than one group, e.g., if it is involved in more than one internal symmetry relation. Each GROUp requires specification of the weights. This allows some restraints to be tighter than others.

Warning: No check is made to see if NCS-related atoms are frozen during the refinement; all must be free, or all must be frozen.

16.1.1 Syntax

NCS RESTraints { <**NCS-restraints-statement**> } **END** is invoked from the main level of X-PLOR.

<**NCS-restraints-statement**>:==

GROUp { <**ncs-restraints-group-statement**> } **END** adds a new group to the restraint NCS database.

INITialize erases the current restraint NCS database. If no restraints are specified, the energy flag NCS should be turned off as well.

<**NCS-restraints-group-statement**>:==

EQUIvalence=<**selection**> adds a new set of NCS-equivalent atoms within the group to the restraint NCS database.

SIGB=<**real**> gives the target deviation σ_{ncs} for B-factor restraints (deviation from the average B of equivalent atoms) in Å^2 (default=2.0 Å^2).

WEIGht=<**real**> gives the effective energy constant for the positional restraints in kcal mole^{-1} Å^{-2} (default: 300.0 kcal mole^{-1} Å^{-2}).

16.1.2 Requirements

The molecular structure and atomic coordinates have to be defined. This facility can be used without crystallographic information. The atom selections are fragile (Section 2.15).

16.1.3 Example

```
remarks  To be called after reading molecular structure and coordinates

                 {* Turn on the energy flag for positional ncs restraints.*}

flags
  include ncs
end

                  {* Restrain only those residues not in lattice contacts.*}

ncs restraints
    group
      equi (segid 1 and not hydrogen and not (resid 58:66 or resid 90:94
            or resid 127:129 or resid 156:165 or resid 325:328) )
      equi (segid 3 and not hydrogen and not (resid 58:66 or resid 90:94
            or resid 127:129 or resid 156:165 or resid 325:328) )
      equi (segid 5 and not hydrogen and not (resid 58:66 or resid 90:94
            or resid 127:129 or resid 156:165 or resid 325:328) )

      weight-ncs=300.
```

```
21        sigb=1.0
22     end
23
24     group
25       equi (segid 2 and not hydrogen and not (resid 10:12 or resid 26:28))
26       equi (segid 4 and not hydrogen and not (resid 10:12 or resid 26:28))
27       equi (segid 6 and not hydrogen and not (resid 10:12 or resid 26:28))
28
29                                {* Put looser restraints on the second group.*}
30       weight-ncs=200.
31       sigb=1.5
32     end
33
34     ?                                {*Print the NCS relations when starting.*}
35  end
36
37
```

16.2 Strict NCS

Coordinate X' is related by NCS to input coordinate X according to

$$X' = \mathcal{R}X + \vec{T} \tag{16.3}$$

where $\mathcal{R}$ is a 3x3 rotation matrix and $\vec{T}$ is a translation vector.

The $\mathcal{R}$ matrices and the $\vec{T}$ vectors are entered using the XNCSrelation and NCSRelation statements. The first 3 input vectors correspond to the ROWS of $\mathcal{R}$, and the 4th vector is $\vec{T}$. It is not necessary to input the identity transformation, since the program sets it by default, but the user may do so for completeness.

The NCS-related interprotomer nonbonded contacts can be turned on by turning on the packing energies PVDW and PELE. Under this option, these energies are for NCS-related molecules only; the lattice energies are automatically shut off.

Many users will have the NCS relations defined in a skew frame (an auxiliary frame) in which the NCS relations have simple forms (e.g., pure rotations). The SKEW option allows a user to specify the relation between this frame and the orthogonal Å frame using a rotation matrix and translation vector. Coordinates in the skew frame are also given in Å. In this case, XNCSrelation and NCSRelation statements take the NCS relations in the skew frame. To use this option, SKEW must be given before XNCSrelation and NCSRelation.

For defining the skew frame, the algebraic conventions of Bricogne's (1976) skewing and averaging method have been adopted, except that the reference frame is the orthogonal frame used by XPLOR. This corresponds to Bricogne's ijk frame. The matrix $\mathcal{P}$ is Bricogne's "transition matrix." The vector $\vec{O}$ gives the coordinates of the origin of the skew frame in the reference frame, and therefore corresponds to Bricogne's $\mathcal{Q}^{-1}\vec{O}'$. Using the following conventions:

$\vec{X}_1$ = input coordinate in reference orthogonal Å frame

$\vec{X}_1'$ = NCS-related coordinate in orthogonal Å frame

$\vec{X}_2$ = input coordinate in skew frame

$\vec{X}_2'$ = NCS-related coordinate in skew frame

$\mathcal{P}$ = skew (transition) matrix

$\vec{O}$ = $\mathcal{Q}^{-1}\vec{O}'$ = origin of skew frame in the reference frame (in Bricogne's package, $\vec{O}'$ is in fractional crystallographic coordinates; $\mathcal{Q}^{-1}$ is the cell orthogonalization matrix)

$\mathcal{U}$ = rotation matrix relating $\vec{X}_2$ and $\vec{X}_2'$ (in skew frame)

$\vec{V}$ = translation vector relating $\vec{X}_2$ and $\vec{X}_2'$

one obtains the following relationships:

$$\vec{X}_1 = \mathcal{P}\vec{X}_2 + \mathcal{Q}^{-1}\vec{O}' \tag{16.4}$$

$$\vec{X}_2 = \mathcal{P}^{-1}\vec{X}_1 - \mathcal{P}^{-1}\mathcal{Q}^{-1}\vec{O}' \tag{16.5}$$

$$\vec{X'}_2 = \mathcal{U}\vec{X}_2 + \vec{V} = \mathcal{U}\mathcal{P}^{-1}\vec{X}_1 - \mathcal{U}\mathcal{P}^{-1}\mathcal{Q}^{-1}\vec{O}' \quad + \quad \vec{V}. \tag{16.6}$$

$$\vec{X'}_1 = \mathcal{P}\vec{X'}_2 + \mathcal{Q}^{-1}\vec{O}', \tag{16.7}$$

$$\vec{X'}_1 = \mathcal{P}\mathcal{U}\mathcal{P}^{-1}\vec{X}_1 - \mathcal{P}\mathcal{U}\mathcal{P}^{-1}\mathcal{Q}^{-1}\vec{O}' + \mathcal{P}\vec{V} + \mathcal{Q}^{-1}\vec{O}' \tag{16.8}$$

$$\mathcal{R} = \mathcal{P}\mathcal{U}\mathcal{P}^{-1} \tag{16.9}$$

$$\vec{T} = -\mathcal{P}\mathcal{U}\mathcal{P}^{-1}\mathcal{Q}^{-1}\vec{O}' + \mathcal{P}\vec{V} + \mathcal{Q}^{-1}\vec{O}' \tag{16.10}$$

Users wishing to convert their Bricogne GENERATE mode 1 input to the XPLOR input should note the following:

1. $\mathcal{P}$ is input as ROWS, not columns as in GENERATE.

2. The skew vector is input after the skew matrix, and is given in orthogonal Å.

3. It is not necessary to input the Identity NCS operation.

4. The NCS rotation matrices are input as rows, not columns.

16.2.1 Syntax

NCS STRIct { <**NCS-strict-statement**> } **END** is invoked from the main level of X-PLOR.

<**NCS-strict-statement**>:==

INITialize erases the strict NCS database.

NCSRelations <**matrix**> [**TRANslation=**<**vector**>] specifies relationships in addition to XNCSrelations, between the input coordinates and their equivalents, to specify completely the internal symmetry for nonbonded interactions. These must be given only after all XNCSrelations have been given. The matrix gives the matrix $\mathcal{R}$, the vector the translation vector $\vec{T}$.

SKEW <**matrix**> [**TRANslation=**<**vector**>] specifies the relationship between the crystal and skew frames. The matrix specifies the matrix $\mathcal{P}$, the vector the translation vector $\vec{O}$ (for the syntax of matrix, see Section 2.4). (default: identity for the matrix and no translation, i.e., no skewing).

XNCSrelations <**matrix**> [**TRANslation=**<**vector**>] specifies the relationships between the input coordinates and their NCS equivalents. The matrix gives the the matrix $\mathcal{R}$, the vector the translation vector $\vec{T}$. (For the syntax of matrix, see Section 2.4.) The XNCSrelations are used to generate the crystallographic asymmetric unit from the protomer. The identity operation is optional.

16.2.2 Requirements

The molecular structure and atomic coordinates have to be defined. This facility can be used without crystallographic information.

16.2.3 Example

In this example, all transformations needed for the nonbonded interactions are also needed for structure factor calculations; hence there are no NCSRelation statements.

```
ncs strict                  {*Invoke strict non-crystallographic symmetry.*}

    skew
      matrix=( .7956571  -.5963627 -.1062134 )
             ( .6057472   .7833304  .1395127 )
             ( .0000000  -.1753427  .9845075 )
      translation=(  22.92660 49.46780   9.72152  )
    end
    xncsrel
       matrix=(-.5  0.  .866025)
              (0.   1.    0.  )
              (-.866025 0. -.5 )
       translation=(0. 0. 0.)
    end
```

```
15     xncsrel
16        matrix=(-.5 0. -.866025)
17               (0.  1.       0.)
18               (.866025 0.  -.5)
19        translation=(0. 0. 0.)
20     end
21    ?                                          {* Check the relations.*}
22 end
```

17 Molecular Replacement

Procedures for molecular replacement in crystallography include self- and cross-rotation functions, translation searches, and the newly developed generalized molecular replacement technique based on Patterson correlation (PC) refinement .

For further reading about PC-refinement, see Brünger (1990), Brünger (1991c), and Brünger et al. (1991).

17.1 Rotation Search

X-PLOR performs conventional rotation searches in Patterson space. The real-space Patterson search method of Huber and Steigemann (1985) is employed. The stationary Patterson map P2 is computed from the observed intensities by fast Fourier transformation on the specified grid. The Patterson map P1 to be rotated is computed from either the observed intensities (self-rotation search) or from a search model (cross-rotation search). Note that the rotation search routine takes the layout of the map P1 as it was written using the xrefin map statement (see Section 14.1). In most cases, the layout of P1 will look like this:

```
EXTEnd=BOX
XMIN=-45.0 XMAX=45.0
YMIN=-45.0 YMAX=45.0
ZMIN=0.0 ZMAX=30.0
```

where one makes use of the centrosymmetry and 30.0 is the maximum vector length. Patterson vectors are selected according to the value of the Patterson map at the particular grid point (THREshold) and according to the distance from the grid point to the origin (RANGe). Of this selected subset of Patterson vectors, only the strongest NPEAKs Patterson vectors are chosen for the rotation search.

The selected Patterson vectors are rotated using Eulerian angles (θ_1, θ_2, θ_3) as defined by Rossmann and Blow (1962), pseudo-orthogonal Eulerian angles as defined by Lattman (1972), or spherical polar angles (ψ, ϕ, κ) (see Section 2.4 for the conventions). The Lattman angles are related to the

Eulerian angles by

$$\begin{aligned} \theta_+ &= \theta_1 + \theta_3 \\ \theta_- &= \theta_1 - \theta_3 \\ \theta_2 &= \theta_2 \end{aligned} \tag{17.1}$$

The θ_2 angles are sampled at a constant interval Δ. Following Lattman (1972), the variable interval for θ_+ is given by $\Delta/\cos(\theta_2/2)$, and the variable interval for θ_- is given by $\Delta/\sin(\theta_2/2)$. For both Eulerian angles and spherical polar angles, the sampling interval is constant (Δ) for the three angular variables.

The rotation search should be restricted to an appropriately chosen asymmetric unit (Rao et al., 1980). Table 17.1 lists asymmetric units for cross-rotation functions for search models without internal symmetry. P_1 is the rotated Patterson function of the search model and P_2 is the observed Patterson function. The Laue group of P_1 is "-1". The Laue group of P_2 is determined by the space group of the crystal.

Laue Group	Crystal System	Euler Angles Lattman Angles
-1	triclinc	$0 \le \theta_1 < 2\pi, 0 \le \theta_2 \le \pi, 0 \le \theta_3 < 2\pi$ $0 \le \theta_- \le 2\pi, 0 \le \theta_2 \le \pi, 0 \le \theta_+ < 4\pi$
2/m (b unique)	monoclinic (b-u)	$0 \le \theta_1 < 2\pi, 0 \le \theta_2 \le \pi/2, 0 \le \theta_3 < 2\pi$ $0 \le \theta_- \le 2\pi, 0 \le \theta_2 \le \pi/2, 0 \le \theta_+ < 4\pi$
2/m (c unique)	monoclinic (c-u)	$0 \le \theta_1 < 2\pi, 0 \le \theta_2 \le \pi, 0 \le \theta_3 < \pi$ $0 \le \theta_- \le \pi, 0 \le \theta_2 \le \pi, 0 \le \theta_+ < 4\pi$
mmm	orthorhombic	$0 \le \theta_1 < 2\pi, 0 \le \theta_2 \le \pi/2, 0 \le \theta_3 < \pi$ $0 \le \theta_- \le \pi, 0 \le \theta_2 \le \pi/2, 0 \le \theta_+ < 4\pi$
4/m	tetragonal	$0 \le \theta_1 < 2\pi, 0 \le \theta_2 \le \pi, 0 \le \theta_3 < \pi/2$ $0 \le \theta_- \le \pi/2, 0 \le \theta_2 \le \pi, 0 \le \theta_+ < 4\pi$
4/mmm	tetragonal	$0 \le \theta_1 < 2\pi, 0 \le \theta_2 \le \pi/2, 0 \le \theta_3 < \pi/2$ $0 \le \theta_- \le \pi/2, 0 \le \theta_2 \le \pi/2, 0 \le \theta_+ < 4\pi$
-3	trigonal	$0 \le \theta_1 < 2\pi, 0 \le \theta_2 \le \pi, 0 \le \theta_3 < 2\pi/3$ $0 \le \theta_- \le 2\pi/3, 0 \le \theta_2 \le \pi, 0 \le \theta_+ < 4\pi$
-3m	trigonal	$0 \le \theta_1 < 2\pi, 0 \le \theta_2 \le \pi/2, 0 \le \theta_3 < 2\pi/3$ $0 \le \theta_- \le 2\pi/3, 0 \le \theta_2 \le \pi/2, 0 \le \theta_+ < 4\pi$
6/m	hexagonal	$0 \le \theta_1 < 2\pi, 0 \le \theta_2 \le \pi, 0 \le \theta_3 < \pi/3$ $0 \le \theta_- \le \pi/3, 0 \le \theta_2 \le \pi, 0 \le \theta_+ < 4\pi$
6/mmm	hexagonal	$0 \le \theta_1 < 2\pi, 0 \le \theta_2 \le \pi/2, 0 \le \theta_3 < \pi/3$ $0 \le \theta_- \le \pi/3, 0 \le \theta_2 \le \pi/2, 0 \le \theta_+ < 4\pi$

Table 17.1: Asymmetric Units of Cross-Rotation Functions.

The values of the Patterson map P2 at the positions of the rotated Patterson vectors of map P1 are computed by linear eight-point interpolation

(Nordman 1980).

For each sampled orientation Ω the product function

$$RF(\Omega) =< P_{obs} P_{model}(\Omega) > \tag{17.2}$$

between the rotated vectors of P1 and the interpolated values of the Patterson map P2 is computed.

Upon completion of the rotation function, all sampled grid points are sorted with respect to their product correlation value, and a simple peak search is carried out using the matrix metric defined in Brünger (1990). For two given rotation matrices Ω^1, Ω^2, the metric is defined as

$$m(\Omega^1, \Omega^2) = \min_{s=1,n_s} \sqrt{\mathrm{Tr}\{(\Omega^1 - \mathcal{O}_s\Omega^2)(\Omega^1 - \mathcal{O}_s\Omega^2)^t\}} \tag{17.3}$$

where n_s is the number of symmetry operators of the space group of the crystal and $\mathcal{O}_s$ is the rotational part of the symmetry operator s. This definition applies to the default option SELF=FALSE. If SELF=TRUE is specified, the metric is defined as

$$m(\Omega^1, \Omega^2) = \min_{s=1,n_s,s'=1,n_s} \sqrt{\mathrm{Tr}\{(\Omega^1 - \mathcal{O}_s\Omega^2\mathcal{O}'_s)(\Omega^1 - \mathcal{O}_s\Omega^2\mathcal{O}'_s)^t\}}, \tag{17.4}$$

i.e., the metric assumes the crystal symmetry for both Patterson maps P1 and P2. In the case of space group P_1, the metric m is related to a rotation angle κ, which defines a rotation around a certain axis a:

$$m = \sqrt{4(1 - \cos\kappa)} \tag{17.5}$$

Application of this rotation around a transforms the matrix Ω^1 into Ω^2.

Two RF grid points are considered as being in the same cluster if the corresponding rotation matrices yield $m(\Omega^1, \Omega^2) < \epsilon$. For example, if ϵ is set to 0.25, matrices belong to the same cluster if they can be transformed by a rotation of 10° or less around a certain axis. The incorporation of crystallographic symmetry in Eq. 17.3 ensures that clusters of grid points at the boundaries of the asymmetric unit of the RF are treated properly.

This peak search removes grid points that are close to grid points with larger RF values. It is not a true peak search, but rather reduces the number of points to be checked by subsequent analysis. The reduced set of the highest grid points is written to a specified file. For example, if ϵ is set to 0.25, the file will contain grid points that are mutually different by at least 10°. This file will be read by the *PC*-refinement procedure described below. The value for ϵ should be chosen to be less than the radius of convergence of the *PC*-refinement (around 10°). It should be noted that the peak search procedure maps the grid points into an asymmetric unit of the rotation function.

17.1.1 Syntax

SEARch Rotation { <**xrefin-search-rotation-statement**> } **END** is an xrefin statement. Action takes place as soon as the END statement is issued.

<**xrefin-search-rotation-statement**> :==

DELTa=<real> specifies an angular grid interval. DELTa should be less than ArcSin$[1/(3d_{min}v_{max})]$, where $1/d_{min}$ is the high-resolution limit (see Section 12.3) and v_{max} is the maximum Patterson vector length of the search model (see Section 17.5.3 for an example). This is a conservative estimate. In many cases, it will be sufficient to double this estimate. Note that the CPU time requirement for the rotation function is roughly an inverse cubic function of DELTa. Thus, one should use the largest possible value for DELTa (default: 0.0).

EPSIlon=<real> specifies ϵ for the peak cluster analysis. Multiple entries produce multiple cluster analysis (default: 0.2).

FORMatted=<logical> should be set to FALSE if the map files are binary files (default: TRUE).

LIST=<filename> specifies the file for the list of rotation function peaks after the cluster analysis (grid points are mapped into an asymmetric unit of the rotation function) (default: OUTPUT).

NLISt=<integer> specifies that the analysis of the rotation function will be restricted to the <integer> highest grid points.

NPEAks=<integer> selects the <integer> strongest Patterson vectors of map P1 that satisfy the range and threshold criteria (default: 5000).

OUTPut=<filename> specifies the file for the complete rotation function.

P1INput=<filename> reads Patterson map P1.

P2INput=<filename> reads Patterson map P2.

PSIMIn=<real> PSIMAx=<real> PHIMIn=<real> PHIMAx=<real> KAPPAMIn=<real> KAPPAMAx=<real> is spherical polar angle sampling (as defined in Section 2.4). Specified are the minimum and maximum values for ψ, ϕ, κ (default: 0).

RANGe=<real>=<real> selects the Patterson vectors of map P1 based on the distance of the vectors from the origin (default: 3–20 Å).

SELF=<logical> is a flag for cluster analysis that determines the symmetry of map P1. If FALSE, map P1 is treated with P_1 symmetry, whereas map P2 is treated with crystallographic symmetry, as in the xrefin symmetry statement. If TRUE, both P1 and P2 are treated with crystallographic symmetry.

T1MIn=<real> T1MAx=<real> T2MIn=<real> T2MAx=<real> T3MIn=<real> T3MAx=<real> is equidistant Eulerian angle sampling (θ_1, θ_2, θ_3, as defined in Section 2.4) using DELTa intervals. Specified are the minimum and maximum values for θ_1, θ_2, θ_3 (default: 0).

THREshold=<real> selects the Patterson vectors of map P1 according to the value of the Patterson map: if this value is larger than the specified threshold, the Patterson vector is selected (default: 500).

TPMIn=<real> TPMAx=<real> TMMIn=<real> TMMAx=<real> T2MIn=<real> T2MAx=<real> is Lattman's (1972) pseudo-orthogonal sampling, using DELTa as the interval for θ_2. Specified are the minimum and maximum values for θ_+, θ_2, and θ_- (default: 0 and none).

17.1.2 Requirements

The space-group operators and unit-cell parameter have to be specified.

17.1.3 The Rotation Function Listing

The following is an example of the LIST file that enumerates the highest grid points after the cluster analysis:

```
! index, theta1, theta2, theta3, RF-function (EPSIlon= 0.25)
     1    20.765  70.000 253.448    1.2082
     6    35.554  75.000  68.657    1.1648
     7     9.555  67.500 257.055    1.1507
     8    34.519  75.000 249.691    1.1493
     9    14.460  70.000  75.923    1.1466
    10   344.274  65.000  87.131    1.1465
```

The first number is the order of the grid point; i.e., 1 is the highest grid point and 6 is the 6th-highest grid point. In this case, the cluster analysis has determined that grid points 2, 3, 4, and 5 are close to the first point, and thus they do not appear in this list. The second, third, and fourth numbers are the angular parameters according to the selected sampling scheme. The fifth number is the value of the rotation function. Note that the cluster analysis maps the angular parameters into an asymmetric unit of the rotation function, thus removing symmetry-related redundancies.

17.1.4 The Rotation Function Output File

The following example file contains a complete rotation function. It is written as a Mathematica script file. The 3-dimensional matrix is written in (θ_2, θ_3, θ_1), (θ_2, θ_-, θ_+), and (ϕ, κ, ψ) levels for Eulerian angles, Lattman's angles, and spherical polar angles, respectively.

```
(* Rotation Function *)
  rnumber=    393;
  rave=  0.032;
  rsigma=  0.048;
  rmax=  0.221;
  rmin= -0.075;
var={ "theta2", "theta-", "theta+" };
min={    0.00000 ,    0.00000 ,    0.00000 };
max={   90.00000 ,  720.00000 ,  360.00000 };
rf={
{
{
    1,    2
},
{
    3,    4
}
},
{
{   3,    5
},
{   6,    7
}
}
};
```

The values specified are the number of grid points (rnumber), the average of the rotation function (rave), σ (rsigma), maximum (rmax), and minimum (rmin) of the rotation function, the meaning of the three dimensions of the matrix RF (var), and the minimum and maximum values of the three angular parameters. The rotation function itself is specified as a Mathematica list. Note that the size of the θ_-, θ_+ sections may vary with different choices of θ_2. (The use of this file by Mathematica is described in Section 17.4.)

17.2 Comparing Orientations of Molecules

X-PLOR provides a facility to compute and compare rotation matrices. Two matrices are stored (primary and secondary matrix).

17.2.1 Syntax

ROTMan { <**rotman-statement**> } **END** is invoked from the main level of X-PLOR.

<**rotman-statement**> :==

ALLPairs computes the distance $\sqrt{\mathrm{Tr}\{(\Omega^1 - \mathcal{O}_s\Omega^2\mathcal{O}'_s)(\Omega^1 - \mathcal{O}_s\Omega^2\mathcal{O}'_s)^t\}}$ between the primary and secondary matrix for all crystallographic symmetry images. Note, that this option may produce large amount of output.'

COPY copies a primary matrix into a secondary matrix.

DISTance computes the metric (Eq. 17.3) between the primary and the secondary matrix, including crystallographic symmetry images. Upon completion of this operation, the Eulerian angles describing the rotation matrix R that converts the primary into the secondary matrix (or one of its symmetry mates) are stored in the symbols \$THETA1, \$THETA2, \$THETA3, the rotation axes in the symbols \$AXIS_X, \$AXIS_Y, \$AXIS_Z, \$AXIS_KAPPA, the spherical polars in the symbols \$PSI, \$PHI, \$KAPPA, the rotation matrix in the symbols \$ROT_1_1, ..., \$ROT_3_3, and the distance is stored in the symbol \$DISTANCE.

INVErse computes the inverse of the primary matrix and stores it in the matrix.

<**matrix**> enters the primary matrix (see Section 2.4).

PRODuct computes the product [primary*secondary] matrix and stores the result in the primary matrix.

SWAP exchanges primary and secondary matrices.

17.2.2 Requirements

There are no requirements, since this is a stand-alone facility.

17.3 Translation Search

X-PLOR performs a translation search by computing the target function

$$1 - E_{XREF}/W_A \tag{17.6}$$

where the target is specified by TARGet in the xrefin statement . (The various targets are described in section 12.3.) In particular, translation searches can be carried out with correlation coefficients between Fs, Es, F^2s, and E^2s, with the standard residual, with the vector (A,B) residual, or with the packing function. The search is carried out over a 3-dimensional grid that can be specified in fractional ("MODE=FRACtional") or orthogonal Å ("MODE=ORTHogonal") coordinates. The search routine computes the structure factors F_{calc} of the translated primary molecule and the symmetry-related molecules by applying appropriate phase-shift operators in reciprocal space to the calculated structure factors of the original (not translated) molecule and its symmetry mates (except for the packing function). In this way, the computation is very fast, and it is also highly vectorized. The

symmetry mates are defined by the space-group operators in the xrefin statement. The partial structure factors F_{part} are kept constant during the translation search. This feature can be used if one wants to translate only a part of the molecule (see also the example file in section 17.5.8).

The asymmetric unit of a translation function is in general larger than the asymmetric unit of the unit cell. For example, in space group $P6_522$, the asymmetric unit for the translation function is $x = 0 \ldots 1$, $y = 0 \ldots 1$, $z = 0 \ldots 0.5$, whereas the crystallographic asymmetric unit is $x = 0 \ldots 1$, $y = 0 \ldots 1$, $z = 0 \ldots 1/12$. The Euclidean normalizer groups (also referred to as Cheshire groups) define the equivalent positions of a search molecule (Hirshfeld, 1968; Koch & Fischer, 1987). The basis vectors of a particular Euclidean normalizer define an asymmetric unit of the translation function. Euclidean normalizers for all space groups are listed in Tables 17.2 and 17.3. For certain space groups the position of the molecule along one or more unit cell axes is arbitrary; in Tables 17.2 and 17.3 this is indicated by the symbol ϵ. The unit cell axes of the Cheshire group do not necessarily coincide with the unit cell axes of the space group (e.g., Cheshire group P_422). In this case, the XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX values have to be chosen large enough to include the unit cell of the Cheshire group; as a consequence, the space sampled by the translation function is larger than the asymmetric unit.

It is sometimes useful to run a translation search on a very coarse lattice that covers the complete unit cell (i.e., XGRId= 0 0.1 1, YGRId= 0. 0.1 1, ZGRId=0 0.1 1). Based on the observed redundancies, one can then deduce other choices for the asymmetric unit of the translation function not listed in Table 17.1.

The output of the routine provides a complete listing of the computed translation function, which is written to the specified file. The format of this file is suitable for Mathematica. In addition, a sorted list of a specified number of highest peaks is written to the standard output. A packing analysis is carried out for each listed peak. The packing value consists of the ratio of the molecular volume to the unit-cell volume; it is expressed as a percentage. The highest value of the translation function is stored in the symbol $TFMAX; the corresponding translation vector (x, y, z) is stored in the symbols $TFMAX_X, $TFMAX_Y, $TFMAX_Z. The packing function value is stored in the symbol $PACKING. The main coordinate set are translated by (x, y, z). Please note that this translation is applied to all atomic coordinates regardless of the selected atoms for the structure factor calculation.

17.3.1 Syntax

SEARch TRANslation { <**xrefin-search-translation-statement**> }
END is an xrefin statement. Action takes place as soon as the END statement is issued.

Symbol	Basis vectors	Space groups
$P\overline{1}$	$\frac{1}{2}a \times \frac{1}{2}b \times \frac{1}{2}c$	$P\overline{1}$
$Z^3\overline{1}$	$\epsilon a \times \epsilon b \times \epsilon c$	$P1$
$P2/m$	$\frac{1}{2}a \times \frac{1}{2}b \times \frac{1}{2}c$	$P2/m$, $P2_1/m$, $C2/m$, $P2/c$, $P2_1/c$, $C2/c$
Z^12/m	$\frac{1}{2}a \times \epsilon b \times \frac{1}{2}c$	$P2$, $P2_1$, $C2$
Z^22/m	$\epsilon a \times \frac{1}{2}b \times \epsilon b$	Pm, Pc, Cm, Cc
$Pmmm$	$\frac{1}{2}a \times \frac{1}{2}b \times \frac{1}{2}c$	$P222$, $P222_1$, $P2_12_12$, $P2_12_12_1$, $C222_1$, $C222$, $I222$, $I2_12_12_1$, $Pmmm$, $Pnnn$, $Pccm$, $Pban$, $Pmma$, $Pnna$, $Pmna$, $Pcca$, $Pbam$, $Pccn$, $Pbcm$, $Pnnm$, $Pmmn$, $Pbcn$, $Pbca$, $Pnma$, $Cmcm$, $Cmca$, $Cmmm$, $Cccm$, $Cmma$, $Ccca$, $Fmmm$, $Immm$, $Ibam$, $Ibca$, $Imma$
$Pnnn$	$\frac{1}{2}a \times \frac{1}{2}b \times \frac{1}{2}c$	$Fddd$
$Immm$	$\frac{1}{2}a \times \frac{1}{2}b \times \frac{1}{2}c$	$F222$
Z^1mmm	$\frac{1}{2}a \times \frac{1}{2}b \times \epsilon c$	$Pmm2$, $Pmc2_1$, $Pcc2$, $Pma2$, $Pca2_1$, $Pnc2$, $Pmn2_1$, $Pba2$, $Pna2_1$, $Pnn2$, $Cmm2$, $Cmc2_1$, $Ccc2$, $Amm2$, $Abm2$, $Ama2$, $Aba2$, $Fmm2$, $Imm2$, $Iba2$, $Ima2$
Z^1ban	$\frac{1}{2}a \times \frac{1}{2}b \times \epsilon c$	$Fdd2$
$P4_222$	$\frac{1}{2}(a-b) \times \frac{1}{2}(a+b) \times \frac{1}{2}c$	$P4_122$, $P4_12_12$, $P4_322$, $P4_32_12$
Z^1422	$\frac{1}{2}(a-b) \times \frac{1}{2}(a+b) \times \epsilon c$	$P4_1$, $P4_3$
$P4/mmm$	$\frac{1}{2}(a-b) \times \frac{1}{2}(a+b) \times \frac{1}{2}c$	$P\overline{4}$, $P4/m$, $P4_2m$, $P4/n$, $P4_2/n$, $I4/m$, $P422$, $P42_12$ $P4_222$, $P4_22_12$, $I422$, $P\overline{4}2m$, $P\overline{4}2c$, $P\overline{4}2_1m$, $P\overline{4}2_1c$, $P\overline{4}m2$, $P\overline{4}c2$, $P\overline{4}b2$, $P\overline{4}n2$, $I\overline{4}2m$, $P4/mmm$ $P4/mcc$, $P4/nbm$, $P4/nnc$, $P4/mbm$, $P4/mnc$, $P4/nmm$, $P4/ncc$, $P4_2/mmc$, $P4_2/mcm$, $P4_2/nbc$, $P4_2/nnm$, $P4_2/mbc$, $P4_2/mnm$, $P4_2/nmc$, $P4_2/ncm$, $I4/mmm$, $I4/mcm$

Table 17.2: Euclidean normalizers for all space groups.

<xrefin-search-translation-statement> :==

Symbol	Basis vectors	Space groups
$P4_2/nnm$	$\frac{1}{2}(a-b) \times \frac{1}{2}(a+b) \times \frac{1}{2}c$	$I4_1/a$, $I4_122$, $I\overline{4}2d$, $I4_1/amd$, $I4_1/acd$
$I4mmm$	$\frac{1}{2}(a-b) \times \frac{1}{2}(a+b) \times \frac{1}{2}c$	$I\overline{4}$, $I\overline{4}m2$, $I\overline{4}c2$
Z^14/mmm	$\frac{1}{2}(a-b) \times \frac{1}{2}(a+b) \times \epsilon c$	$P4$, $P4_2$, $I4$, $P4mm$, $P4bm$, $P4_2cm$, $P4_2nm$, $P4cc$, $P4nc$, $P4_2mc$, $P4_2bc$, $I4mm$, $I4cm$
Z^14/nbm	$\frac{1}{2}(a-b) \times \frac{1}{2}(a+b) \times \epsilon c$	$I4_1$, $I4_1md$, $I4_1cd$
$R\overline{3}m$	$(-a) \times (-b) \times \frac{1}{2}c$	$R\overline{3}$, $R32$, $R\overline{3}m$, $R\overline{3}c$
$Z^1\overline{3}1m$	$\frac{1}{3}(2a+b) \times \frac{1}{3}(-a+b) \times \epsilon c$	$R3$, $R3m$, $R3c$
$P6_222$	$a \times b \times \frac{1}{2}c$	$P3_121$, $P6_122$, $P6_422$
$P6_222$	$\frac{1}{3}(2a+b) \times \frac{1}{3}(-a+b) \times \frac{1}{2}c$	$P3_112$
$P6_422$	$a \times b \times \frac{1}{2}c$	$P3_221$, $P6_522$, $P6_222$
$P6_422$	$\frac{1}{3}(2a+b) \times \frac{1}{3}(-a+b) \times \frac{1}{2}c$	$P3_212$
Z^1622	$a \times b \times \epsilon c$	$P6_1$, $P6_5$, $P6_2$, $P6_4$
Z^1622	$\frac{1}{3}(2a+b) \times \frac{1}{3}(-a+b) \times \epsilon c$	$P3_1$, $P3_2$
$P6/mmm$	$a \times b \times \frac{1}{2}c$	$P\overline{3}$, $P321$, $P\overline{3}1m$, $P\overline{3}1c$, $P\overline{3}m1$, $P\overline{3}c1$, $P6/m$, $P6_3/m$, $P622$, $P6_322$, $P\overline{6}2m$, $P\overline{6}2c$, $P6mmm$, $P6/mcc$, $P6_3/mcm$, $P6_3/mmc$
$P6/mmm$	$\frac{1}{3}(2a+b) \times \frac{1}{3}(-a+b) \times \frac{1}{2}c$	$P312$, $P\overline{6}$, $P\overline{6}m2$, $P\overline{6}c2$
Z^16/mmm	$a \times b \times \epsilon c$	$P31m$, $P31c$, $P6$, $P6_3$, $P6mm$, $P6cc$, $P6_3cm$, $P6_3mc$
Z^16/mmm	$\frac{1}{3}(2a+b) \times \frac{1}{3}(-a+b) \times \epsilon c$	$P3$, $P3m1$, $P3c1$
$Ia\overline{3}$	$a \times b \times c$	$Pa\overline{3}$
$I4_132$	$a \times b \times c$	$P4_332$, $P4_132$
$Pm\overline{3}m$	$\frac{1}{2}a \times \frac{1}{2}b \times \frac{1}{2}c$	$Fm\overline{3}$, $F432$, $Fm\overline{3}m$, $Fm\overline{3}c$
$Pn\overline{3}m$	$\frac{1}{2}a \times \frac{1}{2}b \times \frac{1}{2}c$	$Fd\overline{3}$, $F4_132$, $Fd\overline{3}m$, $Fd\overline{3}c$
$Im\overline{3}m$	$a \times b \times c$	$P23$, $I23$, $Pm\overline{3}$, $Pn\overline{3}$, $Im\overline{3}$, $P432$, $P4_232$, $I432$, $P\overline{4}3m$, $I\overline{4}3m$, $P\overline{4}3n$, $Pm\overline{3}m$, $Pn\overline{3}n$, $Pm\overline{3}n$, $Pn\overline{3}m$, $Im\overline{3}m$
$Im\overline{3}m$	$\frac{1}{2}a \times \frac{1}{2}b \times \frac{1}{2}c$	$F23$, $F\overline{4}3m$, $F\overline{4}3c$
$Ia\overline{3}d$	$a \times b \times c$	$P2_13$, $I2_13$, $Ia\overline{3}$, $I4_132$, $I\overline{4}3d$, $Ia\overline{3}d$

Table 17.3: Euclidean normalizers for all space groups. Cont'd.

MODE=FRACtional|ORTHogonal specifies whether the grid search will be carried out in fractional or orthogonal Å coordinates (default: FRACtional).

NLISt=<integer> specifies the number of highest grid points that will be listed to the standard output (default: 101).

OUTPut=<filename> specifies the filename for the full listing of the translation function (default: none, i.e., no full listing).

XGRId=<real> <real> <real> specifies the starting value, step size, and ending value for the grid search along the x-direction. The starting value and the ending value must be identical if this direction is to be kept fixed. The step size should be less than $1/(3d_{min}a)]$, where $1/d_{min}$ is the high-resolution limit (see Section 12.3) and a is defined by the unit-cell geometry (see Section 12.3). This is a conservative estimate. In some cases it may be sufficient to double this estimate. Note that the CPU time requirement for the translation function is roughly an inverse cubic function of the step size for a three-dimensional translation function. Thus, one should use the largest possible value for the step size (default: 0. 0. 0.).

YGRId=<real> <real> <real> does the same as above for the y-direction.

ZGRId=<real> <real> <real> does the same as above for the z-direction.

17.3.2 Requirements

Required data are the molecular structure, coordinates, observed reflections, space-group operators, and atomic-form factors.

17.3.3 The Translation Function Listing

The following is an example of the list that is produced by the translation function:

```
 List of first    101 highest peaks
Orthogonal A coordinates   Fractional coordinates
T=(-2.  -3.   0.) TF=(  -0.233  -0.217   0.) T= 1.0000 P= 0.2667
T=(-2.   6.   0.) TF=(   0.092   0.433   0.) T= 0.9798 P= 0.2567
T=( 6.   2.   0.) TF=(   0.447   0.144   0.) T= 0.9526 P= 0.2767
T=( 6.  -8.   0.) TF=(   0.086  -0.577   0.) T= 0.9438 P= 0.2533
T=( 6.  -7.   0.) TF=(   0.122  -0.505   0.) T= 0.8834 P= 0.2483
T=( 3.   8.   0.) TF=(   0.476   0.577   0.) T= 0.8418 P= 0.2800
```

The grid points are sorted by value, with the highest peak listed first. The position of the molecule is given in both orthogonal and fractional coordinates. The value of the translation function and the corresponding packing value are listed in the last two columns.

17.3.4 The Translation Function Output File

The following example file contains a complete translation function. It is written as a Mathematica script file. The 3-dimensional matrix is written in

(x, y, z) levels.

```
 (* Translation Function *)
   rnumber=    289;
   rave=  0.688;
   rsigma=  0.088;
   rmax=  1.000;
   rmin=  0.461;
 var={ "x", "y", "z" };
 min={   -8.00000 ,   -8.00000 ,    0.00000 };
 max={    8.00000 ,    8.00000 ,    0.00000 };
 rf={
 {
 {
     1,    2
 },
 {
     3,    4
 }
 },
 {
 {   3,    5
 },
 {   6,    7
 }
 }
 };
```

The values specified are the number of grid points (rnumber), the average of the translation function (rave), σ (rsigma), maximum (rmax), and minimum (rmin) of the translation function, the meaning of the three dimensions of the matrix RF (var), and the minimum and maximum values of the three angular parameters. The translation function itself is specified as a Mathematica list. (The use of this file is described in Section 17.4.)

17.4 A Mathematica Script File

The following Mathematica input file can be used to make contour plots of translation and rotation functions produced by X-PLOR. Note that this procedure can be very time-consuming if a lot of sections are plotted. As always, this example input file can be found in the tutorial subdirectories.

```
1            (* file xtalmr/matrix.math                                    *)
2            (* mathematica file -- Interprets the 3-D matrices written by *)
3            (* XREFIn SEARch ROTAtion and XREFin SEARch TRANslation       *)
4
```

```
Clear[CP];
Off[General::spell1];
$DefaultFont={"Times-Roman",13};

(*===>*)         (* Specify the file name of the matrix written by X-PLOR.*)
  << rotation.3dmatrix ;


       (* With the following statement, one can permute the indices of the *)
       (* 3d-matrix.  This is useful if one wants to plot sections of the  *)
       (* first, second, or third level.  Example:                         *)
       (* perm=(2,1,3); means that one is going to plot 2-sections, where  *)
       (* each 2-section has the 1-coordinate along the x-axis and         *)
       (* the 3-coordinate along the y-axis.                               *)
       (* The meaning of the first, second, and third level depends on the *)
       (* particular application; for example, in the case of spherical    *)
       (* polar angles in XREFin SEARch ROTAtion, it is phi,               *)
       (* kappa, psi.  In this case, one might want to use perm={2,1,3};   *)
       (* one must not permute the indices for matrices with variable      *)
       (* second and third dimensions.  This is the case for Lattman's     *)
       (* angle sampling in XREFin SEARch ROTAtion.                        *)

(*===>*)  perm={1,2,3};

  If[ perm != {1,2,3},
       {iperm=perm;iperm[[perm[[1]]]]=1;
        iperm[[perm[[2]]]]=2;iperm[[perm[[3]]]]=3;
        rf=Transpose[rf,iperm];
        min=min[[perm]];max=max[[perm]];var=var[[perm]];},
     {}];

  nz=Dimensions[rf][[1]];

       (* In the following, one loops through all sections as a function  *)
       (* of the first level.  This can be pretty verbose.  To plot a     *)
       (* particular section near "z", one can compute the index ii by    *)
       (* using the following statement:                                  *)
       (*  ii=Round[(z-min[[1]])/(max[[1]]-min[[1]])*(nz-1)+1].           *)
       (* Of course, "z" should be set before issuing this statement.     *)
       (* Then remove the Do loop and plot the single section.            *)

(*===>*)  Do[ {
   section=Transpose[rf[[ii]]];

               (* Mathematica needs a square matrix to make contour plots.*)
               (* One has to pad the rectangular matrices.                *)
   {ny,nx}=Dimensions[section];
   nmax=Max[nx,ny];
    If[ nz == 1,
       {label=StringJoin[var[[1]],"=",ToString[min[[1]]]]    },
       {label=StringJoin[var[[1]],"=",ToString[(ii-1)/(nz-1)
                        (max[[1]]-min[[1]])+min[[1]] ] ];   }
      ];
   Which[ ny==1,
                                              (* One-x-dimensional plot.*)
   { column0= Table[min[[2]]+(i-1)(max[[2]]-min[[2]])/nx, {i,1,nx}];
     xyp=Transpose[{column0,(Flatten[section]-rave)/rsigma}];
     ListPlot[xyp,
       PlotLabel->label,
       PlotRange->{ {min[[2]],max[[2]]},{ 0 ,(rmax-rave)/rsigma} },
       AxesLabel->{ var[[2]], "sigma" },
       PlotJoined->True,
(*===>*) Ticks->{Range[min[[2]],max[[2]],50.],
                 Range[0,(rmax-rave)/rsigma+0.5,1.]},
      AxesStyle->{PostScript
         ["/Times-Roman findfont 13 scalefont setfont"]}];
              },
       nx==1,
                                              (* One-y-dimensional plot.*)
   { column0= Table[min[[3]]+(i-1)(max[[3]]-min[[3]])/ny, {i,1,ny}];
```

```
75      xyp=Transpose[{column0,(Flatten[section]-rave)/rsigma}];
76      ListPlot[xyp,
77        PlotLabel->label,
78        PlotRange->{ {min[[3]],max[[3]]},{ 0 ,(rmax-rave)/rsigma} },
79        AxesLabel->{ var[[3]], "sigma" },
80        PlotJoined->True,
81  (*===>*) Ticks->{Range[min[[3]],max[[3]],50.],
82                   Range[0,(rmax-rave)/rsigma+0.5,1.]},
83       AxesStyle->{PostScript
84          ["/Times-Roman findfont 13 scalefont setfont"]}];
85               },
86     True,{
87                                                  (* Two-dimensional plot.*)
88    If[ nmax > ny,
89      { xxx=Table[rave, {j, 1, nmax nmax -nx ny}];
90      section=Partition[Flatten[Append[section, xxx]], nmax];
91      pymax=nmax/ny ( max[[3]] - min[[3]] ) + min[[3]];
92      pxmax=max[[2]];
93       } ,
94      { xxx=Table[rave, {j, 1, nmax nmax -nx ny}];
95      section=Transpose[section];
96      section=Partition[Flatten[Append[section, xxx]], nmax];
97      section=Transpose[section];
98      pymax=max[[3]];
99      pxmax=nmax/nx ( max[[2]] - min[[2]] ) + min[[2]];
100      } ];
101     CP[ii]=ListContourPlot[section,
102       MeshRange->{{min[[2]],pxmax},{min[[3]],pymax}},
103       PlotLabel->label,
104
105 (*===>*)                          (* Specify plotrange: levels starting at *)
106                                   (* 2 sigma above the mean.               *)
107       PlotRange->{ rave + 2 rsigma , rmax },
108       Axes->True,
109       AxesLabel->{ var[[2]], var[[3]] },
110 (*===>*) Ticks->{Range[min[[2]],max[[2]],50.],
111                  Range[min[[3]],max[[3]],50.]},
112
113 (*===>*)                               (* Specify number of contour levels.*)
114       Contours->10,
115       ContourShading->False,
116       ContourSmoothing->None];
117  }];
118  },{ii,nz}];
119
```

17.5 Generalized Molecular Replacement

The following sections show how the 26-10 Fab fragment complexed with digoxin was solved by generalized molecular replacement (Brünger 1991c). The space group of the 26-10 Fab/digoxin crystals is $P2_1$ (Strong 1990), with the b-axis unique, and a non-crystallographic twofold symmetry is present. The following strategy was employed to solve the structure:

1. Self-rotation function.
2. Modification of the elbow angle of a known Fab structure.
3. Cross-rotation function with the modified Fab structure.
4. Filtering the rotation function by PC-refinement.

5. Analysis of the *PC*-refinement.
6. Translation function for molecule A, using the *PC*-refined model.
7. Translation function for molecule B.
8. Combined translation function to determine the relative position between A,B.
9. Rigid-body refinement.

In general, one has to try several different starting elbow angles (spaced approximately 10° apart) and repeat steps 3–6 until a good solution is found. This strategy is also applicable to other multidomain proteins or *PC*-refinements of other degrees of freedom. The modification of the Fab example input files should be straightforward. An overview of the strategy is shown in Fig. 17.1.

17.5.1 Self-rotation Function

The following file performs a self-rotation function. Note that all crystallographic symmetry operators of the space group have to be specified without the translational part.

```
remarks  file xtalmr/self_rf_new.inp  -- Self-rotation function
remaks        -- with origin removal -- crystal vs. crystal

{===>}                          {*Specify location of Patterson map files.*}
evaluate ( $p1_map="p1_map.dat" )
evaluate ( $p2_map="p2_map.dat" )

{===>}                                              {* Resolution limits. *}
evaluate ($low_res=10.)
evaluate ($high_res=4.)

{===>}
evaluate ( $max_vector=45. )     {*Maximum Patterson vector to be searched.*}
evaluate ( $m_max_vector=-$max_vector )

xrefin
{===>}                                          {*Unit cell for crystal.*}
   a=44.144 b=164.69 c=70.17 alpha=90. beta=108.50 gamma=90.

{===>}
   evaluate ($sg="P2(1)")            {* Space group P2(1), b axis unique. *}
   symmetry reset
   @SYMMETRY:symlib.sym

   evaluate ($sg=$patt_symm)
   symmetry reset
                             {* Switch to Patterson symmetry space group. *}
   @SYMMETRY:symlib.sym

{===>}
   nreflections=30000
   reflection @3r9a_det.fob end                        {*Read reflections.*}

   resolution $low_res $high_res

   do (fobs=0) ( amplitude(fobs) < 2 * sigma )             {*Sigma cutoff.*}
```

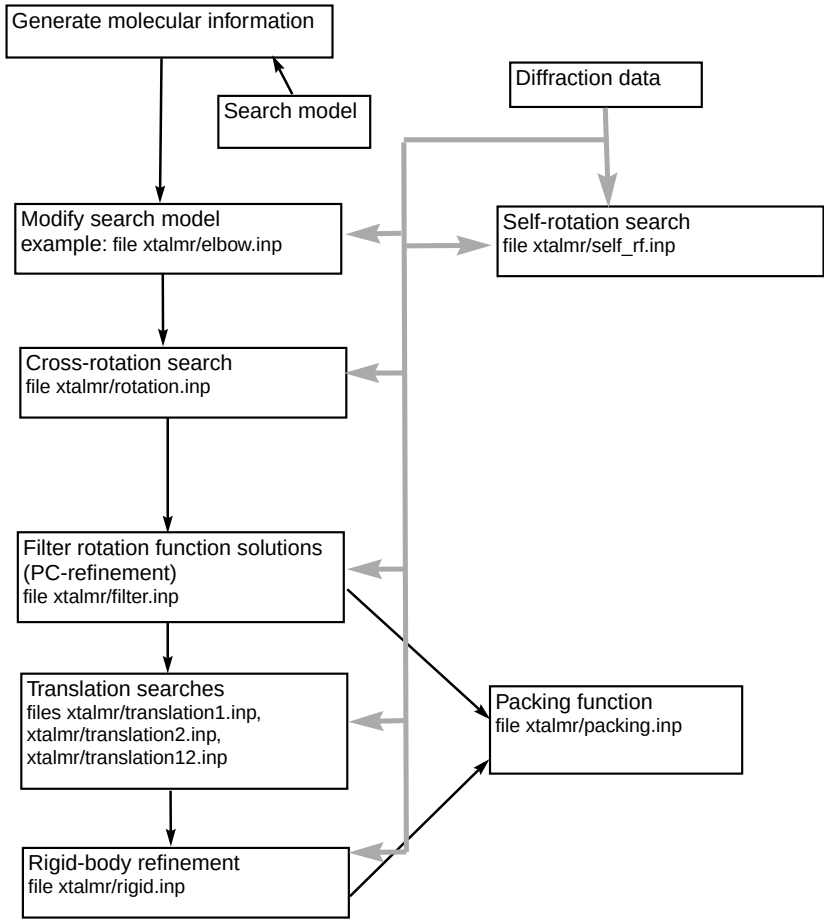

Figure 17.1: Overview of molecular replacement.

```
37
38    fwind=0.1=100000
39
40    method=fft
41    fft
42       grid=0.25       {*Sampling grid for Patterson maps (1/4 high resol.).*}
43    end
44
45                         {* Compute Patterson function with origin removal. *}
46    do (fcalc=combine(amplitude(fobs)^2-sAVE[BINS=10](amplitude(fobs)^2),0))
47                         ( ampl(fobs) > 0 and ( $low_res >= d >= $high_res  ))
48
49    do (map1=ft(fcalc))  ( ampl(fobs) > 0 and ( $low_res >= d >= $high_res  ))
50
51    write map
```

```
      from=map1
                            {*Write a hemisphere of Patterson vectors with*}
      extend=box            {*lengths less than $max_vector.              *}
      xmin=$m_max_vector xmax=$max_vector
      ymin=$m_max_vector ymax=$max_vector
      zmin=0.0 zmax=$max_vector

      automatic=true                        {*Use automatic scaling of map.*}

      formatted=false
      output=$p1_map                       {*Write an unformatted map file.*}
   end


   map                                          {*Compute Patterson map P2.*}
      extend=unit
      automatic=true                        {*Use automatic scaling of map.*}
      formatted=false
      output=$p2_map
   end
end


xrefin
   nrefl=10                                          {*Release some memory.*}
   search rotation
      p1input=$p1_map    formatted=false
      p2input=$p2_map

{===>}
      range=5.0 $max_vector       {*Patterson vector selection for map P1.*}
      threshold=0.0
      npeaks=4000

{===>}
      psimin=0. psimax=180.                {*Spherical polar angular grid.*}
      phimin=0. phimax=180.
      kappamin=160.  kappamax=180.
      delta=2.5
         {*Delta should be less than ArcSin[ high resol / (3*$max_vector)].*}
               {*This is a conservative estimate.  In most cases it will be*}
               {*sufficient to double this estimate.                        *}

      nlist=6000         {*Analyze highest 6000 peaks of rotation function.*}
      selfsymmetry=true           {*Use crystal symmetry of both P1 and P2*}
                                  {*for analysis.                          *}

      epsilon=0.35                     {*Matrix norm for cluster analysis.*}

      list=self_rf_new.rf                  {*Output file for cluster analysis.*}

      output=self_rf_new.3dmatrix         {*Output file for rotation function;*}
                                     {*first level is phi, second kappa, *}
                                     {*third psi.                        *}
  end
end


stop
```

X-PLOR produced the following list of highest rotation function peaks:

```
 ! index, psi, phi, kappa, RF-function ( 0.35)
      1     0.000   0.000 180.000    4.9837
    123    90.000   0.000 180.000    4.4721
   1399   174.000   0.000 170.000    2.5059
```

```
   1481     90.000  84.000 170.000     2.4751
   1492      6.000   0.000 170.000     2.4690
...
```

cleary indicating the presence of a non-crystallographic symmetry twofold element at (ψ=90, ϕ=0, κ=180). This twofold element is visible in the contour plot shown in Fig. 17.2 as well:

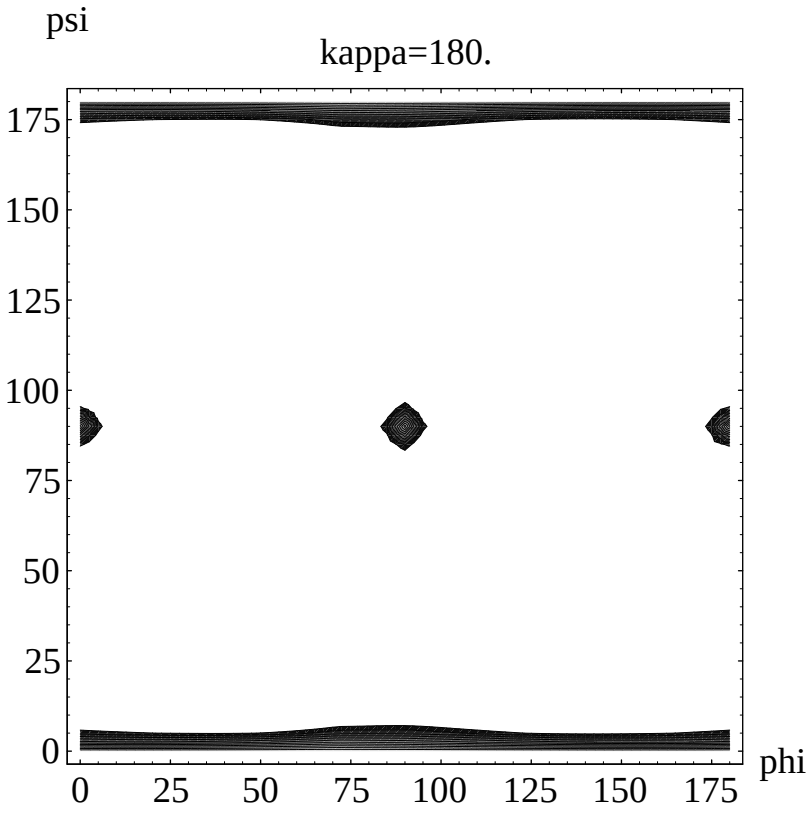

Figure 17.2: Self-rotation function.

17.5.2 Modification of the Elbow Angle of a Known Fab Structure

The following file shows how to modify the elbow angle of a known Fab structure. It should be noted that the definition of the angle is somewhat dependent on the particular Fab sequence.

```
1 remarks file xtalmr/elbow.inp  -- Modify the elbow angle of the search model
2
3 {===>} structure @fab2hfl.psf end                    {*Read structure file.*}
4
5 {===>} coor @fab2hfl.pdb                               {*Read coordinates.*}
6
7
8       {*The following statement rotates the variable domains around an axis*}
```

```
      {*that goes through the two linker regions between the variable and  *}
      {*constant domains.                                                  *}
coor rotate
  vector=(
                                              {*Linker region, light chain.*}
     head=( resid 106 and name ca and segid L )
     tail=( resid 116 and name ca and segid H  )
                                              {*Linker region, heavy chain.*}
         )
  center=( head=( resid 106 and name ca and segid L ) )

  selection=
         ( ( resid 1:106 and segid L ) or ( resid 1:116 and segid H ) )

{===>}
  angle=10
                {*A positive value here will make the elbow angle greater.*}

end

write coordinates output=fab2hfl_10.pdb end            {*Write coordinates.*}

stop

```

17.5.3 Cross-Rotation Function with the Modified Fab Structure

The next step is a cross-rotation function with the modified Fab structure. The model Patterson map P1 is computed by placing the search model into a $120 \times 120 \times 150$ Å orthorhombic box, with the largest extent of the model approximately parallel to the z-direction, and evaluating the structure factors and FFT of the squared amplitudes. Note that all crystallographic symmetry operators of the space group have to be specified without the translational part. The asymmetric unit of the rotation function can be obtained from Table 17.1.

```
remarks file xtalmr/rotation_new.inp  -- Cross-rotation function
remarks        -- with origin removal -- model P1 vs crystal

{===>} structure @fab2hfl.psf end                  {* Read structure file.*}

{===>} coor @fab2hfl.pdb                             {* Read coordinates.*}


{===>}                     {* Select atoms to be included in search model. *}
vector ident (store1) (known and not hydrogen)

                       {* Specify location of temporary Patterson map files.*}
evaluate ( $p1_map="p1_map.dat" )
evaluate ( $p2_map="p2_map.dat" )

{===>}                                              {* Resolution limits. *}
evaluate ($low_res=10.)
evaluate ($high_res=4.)

                                    {* Compute center of geometric center. *}
vector show ave (x) ( store1 )
evaluate ($xcent=$result)
vector show ave (y) ( store1 )
evaluate ($ycent=$result)
vector show ave (z) ( store1 )
```

```
evaluate ($zcent=$result)


       {* Compute maximum distance between geometric center and any atom. *}
vector show max  (sqrt ((x-$xcent)^2 + (y-$ycent)^2 + (z-$zcent)^2) ) (known)

         {* Set the outer radius of the integration volume to this value. *}
evaluate ( $max_vector = $result )

                                                    {* Set the P1 box size *}
evaluate ($box_size=3 * $max_vector)



xrefin                          {* Make Patterson P1 map of model in P1 box.*}

   a=$box_size b=$box_size c=$box_size alpha=90.0 beta=90.0 gamma=90.0

   evaluate ($sg="P1")                                    {* Space group P1. *}
   @SYMMETRY:spacegroup.lib

   @SYMMETRY:scatter.lib

{===>}      {* Allocate sufficient space for the reflections of the P1 box.*}
   nreflections=200000

   resolution $low_res $high_res            {* Resolution range for P1 box.*}

   generate                                {* Generate reflections for P1 box.*}

   method=fft
   fft
      grid=0.25{*Sampling grid for FFT and Patterson map (1/4 high resol.).*}
   end

   selection=(store1)

   update                              {* Compute Fcalcs for model in P1 box.*}

                                    {* Get corresponding Patterson symmetry. *}
   evaluate ($sg=$patt_symm)
   symmetry reset
   @SYMMETRY:symlib.sym

                           {* Compute Patterson function with origin removal. *}
   do (fcalc=combine(
               amplitude(fcalc)^2-sAVE[BINS=10](amplitude(fcalc)^2) ,
                   0))  ( $low_res >= d >= $high_res )

   do (map1=ft(fcalc)) ( $low_res >= d >= $high_res )

   write map
      from=map1
                           {* Write a hemisphere of Patterson vectors with *}
      extend=box           {* lengths less than $max_vector.              *}

      evaluate ( $m_max_vector=-$max_vector )
      xmin=$m_max_vector xmax=$max_vector
      ymin=$m_max_vector ymax=$max_vector
      zmin=0.0 zmax=$max_vector

      automatic=true                        { *Use automatic scaling of map.*}

      formatted=false
      output=$p1_map                       {* Write an unformatted map file.*}
   end
end
```

```
xrefin                                  {* Make Patterson map P2 of crystal.*}

{===>}                                              {* Unit cell for crystal.*}
   a=44.144 b=164.69 c=70.17 alpha=90. beta=108.50 gamma=90.

{===>}
   evaluate ($sg="P2(1)")             {* Space group P2(1), b axis unique. *}
   symmetry reset
   @SYMMETRY:symlib.sym

   evaluate ($sg=$patt_symm)
   symmetry reset
                            {* Switch to Patterson symmetry space group. *}
   @SYMMETRY:symlib.sym



{===>}
   nreflections=30000
   reflection
       reset
       @3r9a_det.fob                                    {* Read reflections.*}
   end

   resolution $low_res $high_res

   do (fobs=0) ( amplitude(fobs) < 2 * sigma )              {*Sigma cutoff.*}

   method=fft
   fft
      grid=0.25      {* Sampling grid for Patterson maps (1/4 high resol.).*}
   end

                         {* Compute Patterson function with origin removal. *}
   do (fcalc=combine(amplitude(fobs)^2-sAVE[BINS=10](amplitude(fobs)^2) ,0))
                        ( ampl(fobs) > 0 and ( $low_res >= d >= $high_res  ))

   do (map1=ft(fcalc)) ( ampl(fobs) > 0 and ( $low_res >= d >= $high_res  ))

   write map
      from=map1
      extend=unit
      automatic=true                       {* Use automatic scaling of map.*}
      formatted=false
      output=$p2_map
   end
end


set abort=off end
xrefin
   nrefl=10                                          {* Release some memory.*}


   search rotation
      p1input=$p1_map    formatted=false
      p2input=$p2_map


      range=5.0 $max_vector       {* Patterson vector selection for map P1.*}
      threshold=0.0
      npeaks=3000                    {* Use 3000 largest vectors of map P1.*}


      @SYMMETRY:cross-rf-asu.lib {* Automatic assignment of asymmtric unit.*}
      tmmin=$tmmin tmmax=$tmmax
      t2min=$t2min t2max=$t2max
      tpmin=$tpmin tpmax=$tpmax

{===>}
```

```
166       delta=2.5
167         {* Delta should be less than ArcSin[ high resol / (3*$max_vector)].*}
168         {* This is a conservative estimate.  In many cases it will be      *}
169         {* sufficient to double this estimate.                              *}
170
171 {===>}
172       list=rotation_new.rf             {* Output file for cluster analysis.*}
173
174
175       nlist=6000        {* Analyze highest 6000 peaks of rotation function.*}
176
177       epsilon=0.25                     {* Matrix norm for cluster analysis.*}
178   end
179 end
180
181 stop
```

X-PLOR produced the following list of highest rotation function peaks:

```
 ! index, theta1, theta2, theta3, RF-function (EPSIlon= 0.25)
      1    14.843  70.000 256.306    4.7813
      2    14.460  70.000  75.923    4.7643
      5    20.250  77.500  92.250    4.6747
      6     3.549  67.500 260.049    4.6676
      8    74.504  82.500 262.163    4.6273
      9     3.173  67.500  79.673    4.6189
     10    20.419  80.000 270.854    4.6173
 ...
```

You may notice that the first two peaks are related by non-crystallographic symmetry. However, this is only a necessary but not a sufficient condition for the correctness of the solution. In fact, these peaks represent incorrect orientations.

17.5.4 PC-Refinement of the Highest Peaks of the Cross-Rotation Function

The next step is to carry out PC-refinement of the highest peaks of the rotation function. PC-refinement (Brünger 1990) of the overall orientation (Ω) and the individual orientations (Ω_i) and translations (t_i) of the four domains of the Fab can be carried out by minimization against a target function defined as

$$E_{TOTAL}(\Omega, \Omega_i, t_i) = (1 - PC(\Omega, \Omega_i, t_i)) \tag{17.7}$$

where

$$PC(\Omega, \Omega_i, t_i) = \tag{17.8}$$

$$\frac{< |E_{obs}|^2 |E_m(\Omega, \Omega_i, t_i)|^2 - \ < |E_{obs}|^2 >< |E_m(\Omega, \Omega_i, t_i)|^2 > >}{\sqrt{< |E_{obs}|^4 - < |E_{obs}|^2 >^2 >< |E_m(\Omega, \Omega_i, t_i)|^4 - < |E_m(\Omega, \Omega_i, t_i)|^2 >^2 >}}.$$

The angle brackets denote an averaging over the set of observed reflections expanded to P_1. E_{obs} denotes the normalized observed structure factors, and $E_m(\Omega, \Omega_i, t_i)$ denotes the normalized structure factors of the search model placed in a triclinic unit cell identical in geometry to that of the crystal.

In the following example, PC-refinements of the search model are carried out where the model is oriented according to each of the 240 selected peaks of the rotation function listed in the file "rotation.rf".

```
remarks file xtalmr/filter.inp   -- PC-refinement of rotation function peaks

{===>} structure @fab2hfl.psf end                    {*Read structure file.*}

{===>} coor @fab2hfl_10.pdb                             {*Read coordinates.*}

evaluate ($wa=10000.)                {* Use this weight if the only active *}
                                     {* energy term is XREF, otherwise, use*}
                                     {* the weight obtained from the       *}
                                     {* "check.inp" protocol.              *}


xrefin
{===>}                                              {*Unit cell for crystal.*}
   a=44.144 b=164.69 c=70.17 alpha=90. beta=108.50 gamma=90.

{===>}
   symmetry=(x,y,z)                 {*Operators for crystal symmetry P2(1).*}
   symmetry=(-x,y+1/2,-z)


   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=40000
   reflection @3r9a_det.fob end                          {*Read reflections.*}

{===>}
   resolution 15.0 4.0                                   {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000


{===>}
   method=fft
   fft
       memory=1000000                  {*FFT method with memory statement.*}
   end

   wa=$wa
   target=E2E2                                          {*Specify target.*}
   mbins=20                        {*Number of bins used for E calculation.*}

   tolerance=0. lookup=false             {*This makes the minimizer happy.*}
```

```
58
59                                        {*Expand data to a P1 hemisphere. *}
60    expand
61 end
62
63 flags exclude * include xref end              {*Use only XREF energy term.*}
64
65 {===>} set display=filter.list end    {*Write the results of the refinement*}
66                                        {*to a file called "filter.list".    *}
67
68 set precision=5 end
69 set message=off end                   {*Turn off messages and echo to reduce*}
70 set echo=off end                      {*output.                            *}
71
72 evaluate ($number=0)
73 evaluate ($counter=0)
74                                    {*Loop over all orientations as specified*}
75                                    {*in file rotation.rf (conventional rf). *}
76
77 for $1 in ( @rotation.rf ) loop main
78
79    evaluate ($counter=$counter+1)              {*This series of statements*}
80    if     ($counter=1) then  evaluate($index=$1){*assigns the information  *}
81    elseif ($counter=2) then  evaluate($t1=$1)   {*of a single line in file *}
82    elseif ($counter=3) then  evaluate($t2=$1)   {*rotation.rf to the       *}
83    elseif ($counter=4) then  evaluate($t3=$1)   {*approp. variables.  A    *}
84    elseif ($counter=5) then                     {*single line contains     *}
85      evaluate ($rf=$1)                          {*$index $t1 $2 $t3 $rf.   *}
86      evaluate ($counter=0)
87
88      evaluate ($number=$number+1)
89
90      coor copy end                          {*Save current coordinates    *}
91      coor rotate euler=( $t1 $t2 $t3 ) end  {*and then rotate them        *}
92                                             {*according to the orientation*}
93                                             {*specified by $t1, $t2, $t3. *}
94
95      energy end                                   {*Compute initial energy*}
96      evaluate ($pc1=1.0-$xref/$wa)                {*and store in $pc1.    *}
97
98      minimize rigid                        {*Rigid-body minimization of the *}
99         drop=10.                           {*orientation of the molecule.   *}
100        nstep=10
101        translation=false            {* Overall translation is kept fixed. *}
102     end
103     evaluate ($pc2=1.0-$xref/$wa)
104
105                                          {*Rigid-body minimization of two*}
106 {===>}                                   {*domains.                       *}
107     minimize rigid
108        group=( ( resid 1:106 and segid L ) or ( resid 1:116 and segid H ) )
109        group=( ( resid 107:212 and segid L )
110        or ( resid 117:213 and segid H ))
111        nstep=15
112        drop=10.
113        translation=false  {* Translation of the last group is kept fixed. *}
114     end
115                                    {*The two domains are further divided   *}
116                                    {*for rigid-body minimization of four   *}
117                                    {*domains: the constant and variable    *}
118 {===>}                             {*domains of the heavy and light chains.*}
119     minimize rigid
120        group=( resid 1:106 and segid L )      {*VL: variable, light chain.*}
121        group=( resid 107:212 and segid L )    {*CL: constant, light chain.*}
122        group=( resid 1:116 and segid H )      {*VH: variable, heavy chain.*}
123        group=( resid 117:213 and segid H )    {*CH1:constant, heavy chain.*}
124        nstep=25
125        drop=10.
126        translation=false  {* Translation of the last group is kept fixed. *}
127     end
```

```
128     evaluate ($pc3=1.0-$xref/$wa)
129
130
131     coor swap end                {*Fit coordinates to starting structure in*}
132     vector do (vx=x) ( all )     {*order to measure the orientation of the *}
133     vector do (vy=y) ( all )     {*PC-refined structure.                   *}
134     vector do (vz=z) ( all )  {*The arrays vx, vy, vz are used as temporary*}
135     coor fit end              {*stores in order to keep the starting       *}
136     vector do (x=vx) ( all )  {*coordinates.                               *}
137     vector do (y=vy) ( all )    {*The COOR FIT statement stores the angles *}
138     vector do (z=vz) ( all )    {*in the symbols $theta1, $theta2, $theta3.*}
139
140
141                           {*Print information: orientation of rotation      *}
142                           {*function peak ($t1, $t2, $t3), orientation      *}
143                           {*after PC-refinement ($theta1, $theta2, $theta3),*}
144                           {*order of the rotation function, rotation        *}
145                           {*function value, and PCs for initial,           *}
146                           {*rigid-body, and domain-refined structures.      *}
147
148     display $t1[F6.2] $t2[F6.2] $t3[F6.2] $theta1[F6.2] $theta2[F6.2] \
149 $theta3[F6.2] $index[F6.0] $rf[F7.2] $pc1[F7.4] $pc2[F7.4] $pc3[F7.4]
150
151    end if
152 end loop main
153
154              {* To sort the refined PC values we can use system utilities. *}
155
156       {* UNIX operating systems: *}
157 system  sort -o filter.sort +10 -r -n filter.list
158
159       {* VMS operating systems: *}
160 !system sort/key=(position:75,size:6,decimal,desc) filter.list filter.sort
161
162
163
164 stop
```

This file produces a new listing stored in the file "filter.list". It contains the results of the PC-refinement. The following shows a few lines of this file:

```
 14.84  70.00 256.31  18.91  68.91 252.84     1.    4.80  0.0211  0.0317  0.0576
 14.46  70.00  75.92  16.81  69.37  75.38     3.    4.75  0.0171  0.0295  0.0466
 20.25  77.50  92.25  18.02  82.52  90.82     4.    4.70 -0.0025  0.0033  0.0559
  3.55  67.50 260.05   6.61  67.67 255.93     6.    4.66 -0.0007  0.0131  0.0416
333.48  82.50  99.86 333.83  83.05 102.35     8.    4.63  0.0098  0.0174  0.0542
```

It lists the orientations before and after PC-refinement and the PC correlation coefficients at various stages of the refinement process. The final PC values (last column) can be sorted as suggested in the bottom of the file "filter.list" using a system sorting procedure. The first six lines of the sorted files are:

```
274.45  35.00 324.68 275.06  34.90 324.70  3402.    3.23  0.0595  0.0683  0.1367
307.70  22.50 294.85 307.57  25.21 291.61  2691.    3.27  0.0318  0.0470  0.0849
291.59  32.50 300.59 292.24  31.90 303.41  3716.    3.22  0.0359  0.0492  0.0811
285.35  30.00 139.40 293.63  30.29 125.31  5975.    3.14  0.0337  0.0431  0.0802
355.89  25.00 251.37   2.38  26.56 248.97  1606.    3.36  0.0307  0.0334  0.0789
```

The final PC values can be plotted by using the following Mathematica script:

```
1               (* file  xtalmr/filter.math                                  *)
2               (* These are Mathematica instructions to make a plot of the *)
```

```
3                (* PCs of the refined structures as a function of rotation  *)
4                (* function peak index.                                       *)
5
6  Off[General::spell1];
7  $DefaultFont={"Times-Roman",13};
8
9                                  (* Change the name of the file "filter.list".*)
10 (*===>*) data = ReadList["filter.sort",
11     {Number,Number,Number,Number,
12                  Number,Number,Number,Number,Number,Number,Number}];
13
14 column11= Transpose[data][[11]];
15 number=Dimensions[column11][[1]];
16 max=Max[column11];
17 ListPlot[column11,
18    AxesLabel->{"Orientations","Refined PC"},
19    Ticks->{Range[0, number , 25],Range[0.0, max ,0.02]},
20    PlotLabel->"PC-Refinement", AxesOrigin->{0,0},PlotJoined->True,
21    PlotRange->{{0, number +2 }, {0., max +0.02 }}];
```

The result of the filtering procedure is shown in Fig. 17.3.

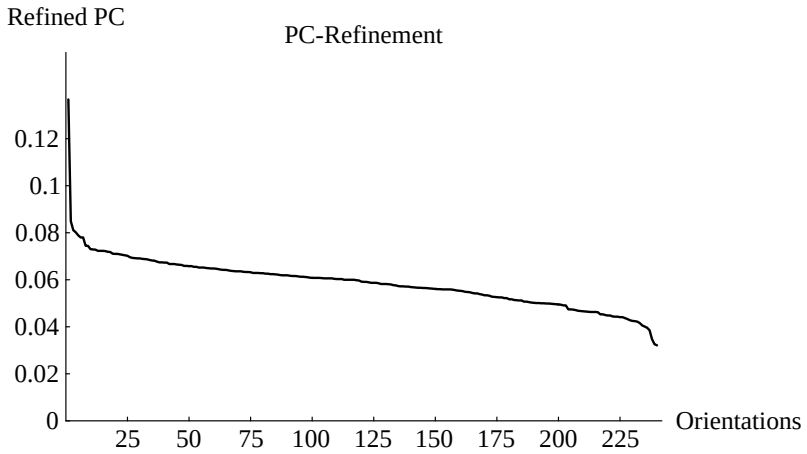

Figure 17.3: Rotation function after PC-refinement.

A single peak is produced. It corresponds to the orientation (θ_1=274.45, θ_2=35, θ_3=324.68) before and (θ_1=275.06, θ_2=34.905, θ_3=324.7) after PC-refinement.

17.5.5 Analysis of the PC-refinement

In the present case, the analysis of the filtering procedure is straightforward: a single peak is produced, which corresponds to the orientation of one of the molecules. The second molecule is not found in this particular run, since its orientation was not among the first 240 peaks of the rotation function. This shows a problem with the PC-refinement strategy: if no solution is found, one cannot be sure that there is no solution. A "direct" rotation search (see Section 17.6) may help in these cases. Another aspect of the

result of the filtering procedure is that sometimes several *PC*-refinements converge on the same solution. The Rotman statement provides a facility to measure the "distance" or metric between two rotation matrices by removing crystallographic redundancies. Suppose one wants to know the distance between two orientations of the search model (θ_1=80.477, θ_2=85.000, θ_3=24.806) and (θ_1=261.392, θ_2=90.000, θ_3= 336.243) taking into account the crystallographic symmetry. Using the Rotman statement

```
xrefin
  a=44.144 b=164.69 c=70.17 alpha=90. beta=108.50 gamma=90.
  symmetry ( x, y, z  )
  symmetry ( -x, y+1/2, -z )
end
rotman
   euler=(80.477  85.000  24.806)
   swap
   euler=( 261.392  90.000 336.243)
   dist
end
```

one finds that the difference between these two orientations is about 5°, due to the crystallographic symmetry.

17.5.6 Translation Function for Molecule A Using the PC-refined Model

Using the structure corresponding to the highest peak after *PC*-refinement, a translation search can be carried out. Because of the low symmetry of the space group in the case of the 26-10 crystal, the search can be restricted to the x, z dimensions.

```
remarks file xtalmr/translation1.inp   -- PC-refinement followed by
remarks translation search

             {*The first part of this job is similar to the PC-refinement  *}
             {*job (filter.inp).  We actually have to repeat the refinement*}
             {*for the selected orientation, since we did not store the    *}
             {*refined coordinates.                                        *}

{===>} structure @fab2hfl.psf end                    {*Read structure file.*}

{===>} parameter @TOPPAR:parhcsdx.pro end            {*Read parameter file.*}

{===>} coor @fab2hfl_10.pdb                             {*Read coordinates.*}

evaluate ($wa=10000.)                                        {* Arbitrary. *}

xrefin
{===>}                                              {*Unit cell for crystal.*}
   a=44.144 b=164.69 c=70.17 alpha=90. beta=108.50 gamma=90.

{===>}
   symmetry=(x,y,z)                {*Operators for crystal symmetry P2(1).*}
   symmetry=(-x,y+1/2,-z)


   SCATter ( chemical C* )
   2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
   12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
```

```
   SCATter ( chemical O* )
   3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
   6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
   6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
   11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=60000
   reflection @3r9a_det.fob end                          {*Read reflections.*}

{===>}
   resolution 15.0 4.0                                   {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000


{===>}
   method=fft
   fft
       memory=2000000                  {*FFT method with memory statement.*}
   end

   wa=$wa
   target=E2E2                {*Specify target used for both PC-refinement*}
                              {*and translation search.                   *}
   mbins=20                       {*Number of bins used for E calculation.*}

   tolerance=0. lookup=false             {*This makes the minimizer happy.*}

                                         {*Expand data to a P1 hemisphere.*}
   expand
end

flags exclude * include xref end              {*Use only XREF energy term.*}

{===>}
coor rotate euler=( 274.45 35 324.68   ) end
                          {*Rotate the structure according to the selected *}
                          {*orientation.  Note: use the orientation that   *}
                          {*comes out of the rotation function (first three*}
                          {*numbers in file "filter.list").                *}

{===>}                     {*Repeat the refinement steps of job filter.inp.*}
minimize rigid
        nstep=10
        drop=10.
        translation=false  {* Translation of the last group is kept fixed. *}
end
minimize rigid
        group=( ( resid 1:106 and segid L ) or ( resid 1:116 and segid H ) )
        group=( ( resid 107:212 and segid L )
        or ( resid 117:213 and segid H ))
        nstep=15
        drop=10.
        translation=false  {* Translation of the last group is kept fixed. *}
end
minimize rigid
        group=( resid 1:106 and segid L )      {*VL: variable, light chain.*}
        group=( resid 107:212 and segid L )    {*CL: constant, light chain.*}
        group=( resid 1:116 and segid H )      {*VH: variable, heavy chain.*}
        group=( resid 117:213 and segid H )    {*CH1:constant, heavy chain.*}
        nstep=25
        drop=10.
        translation=false  {* Translation of the last group is kept fixed. *}
end
```

```
100
101
102                                        {*Turn the crystal symmetry on to  *}
103                                        {*carry out the translation search.*}
104 xrefin
105 {===>}
106    symmetry=(x,y,z)                  {*Operators for crystal symmetry P2(1).*}
107    symmetry=(-x,y+1/2,-z)
108    reduce
109 end
110
111
112                                     {*Get ready for the translation search.*}
113 xrefin
114                          {*Set the grid size for the translation search;    *}
115                          {*it should be less than 1/3 high-resolution limit.*}
116                                 {*This is a very conservative estimate.  In *}
117                                 {*many cases it will be sufficient to double*}
118                                 {*this estimate.                            *}
119
120    evaluate ( $gridx=2. /44. )
121    evaluate ( $gridz=2. /80. )
122    evaluate ( $grid=min($gridx,$gridz) )
123    search translation
124
125       packing=true                           {* Packing analysis turned on. *}
126
127       mode=fractional
128       xgrid=0.0 $grid 0.5             {*We have to search only in x,z in this*}
129       ygrid=0. 0. 0.                  {*space group.  In general, we have to *}
130       zgrid=0. $grid 0.5              {*specify an asymmetric unit for the   *}
131                                       {*translation function.  The basis     *}
132                                       {* vectors of the Eucledian normalizer *}
133                                       {* of the space group provides an      *}
134                                       {* asymmetric unit.                    *}
135
136
137       nlist=100                       {*List the 100 best grid points;       *}
138                                       {*the list is returned in the standard *}
139                                       {*output file.                         *}
140
141       output=translation1.3dmatrix              {*Output matrix for plotting.*}
142                                                 {*This can be verbose for 3d *}
143                                                 {*translation functions!     *}
144
145    end
146 end
147
148
149 write coordinates output=translation1.pdb end  {*The translation function  *}
150                                                 {*returns the coordinates of*}
151                                                 {*best solution.            *}
152
153 xrefin
154    resolution 6. 3.5                    {*Analyze the R-factor distribution*}
155    target=residual                      {*of the best solution.            *}
156    update
157    print r
158 end
159
160 stop
161
```

The translation function clearly identifies a solution that is 9 σ above the next peak. The "matrix.math" script file (see Section 17.4) was used to produce the plot shown in Fig. 17.4.

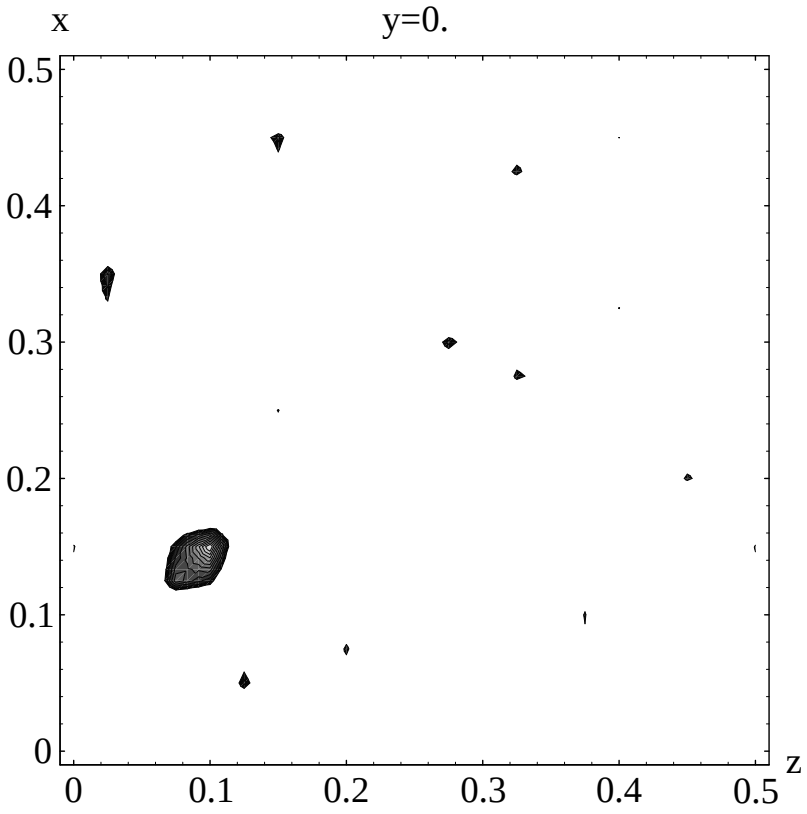

Figure 17.4: Translation function.

17.5.7 Translation Function for Molecule B

Now that we know the orientation and position of molecule A, we can determine the position of molecule B by applying the non-crystallographic symmetry to molecule A and then carrying out another 2-dimensional translation search.

```
remarks file xtalmr/translation2.inp   -- PC-refinement followed by
remarks translation search

                                {*This job is similar to translation1.inp.*}

{===>} structure @fab2hfl.psf end                    {*Read structure file.*}

{===>} parameter @TOPPAR:parhcsdx.pro end            {*Read parameter file.*}

{===>} coor @translation1.pdb                           {*Read coordinates.*}

evaluate ($wa=10000.)                                        {* Arbitrary. *}

xrefin
{===>}                                              {*Unit cell for crystal.*}
   a=44.144 b=164.69 c=70.17 alpha=90. beta=108.50 gamma=90.

{===>}
   symmetry=(x,y,z)                 {*Operators for crystal symmetry P2(1).*}
   symmetry=(-x,y+1/2,-z)
```

```
21
22
23    SCATter ( chemical C* )
24        2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
25    SCATter ( chemical N* )
26       12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
27    SCATter ( chemical O* )
28       3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
29    SCATter ( chemical S* )
30       6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
31    SCATter ( chemical P* )
32       6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
33    SCATter ( chemical FE* )
34       11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070
35
36 {===>}
37    nreflections=60000
38    reflection @3r9a_det.fob end                          {*Read reflections.*}
39
40 {===>}
41    resolution 15.0 4.0                                   {*Resolution range.*}
42
43    reduce
44    do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
45    fwind=0.1=100000
46
47
48 {===>}
49    method=fft
50    fft
51        memory=2000000                  {*FFT method with memory statement.*}
52    end
53
54    wa=$wa
55    target=E2E2                {*Specify target used for both PC-refinement*}
56                               {*and translation search.                    *}
57    mbins=20                       {*Number of bins used for E calculation.*}
58
59    tolerance=0. lookup=false             {*This makes the minimizer happy.*}
60
61                                          {*Expand data to a P1 hemisphere.*}
62    expand
63 end
64
65 flags exclude * include xref end              {*Use only XREF energy term.*}
66
67 {===>}
68 coor rotate spherical=( 90.000   0.000 180.000 ) end
69                                {*Rotate the structure according to the NCS*}
70                                {*symmetry.                                 *}
71
72 {===>}                     {*Repeat the refinement steps of job filter.inp.*}
73 minimize rigid
74         nstep=15
75         drop=10.
76         translation=false  {* Translation of the last group is kept fixed. *}
77 end
78 minimize rigid
79         group=( ( resid 1:106 and segid L ) or ( resid 1:116 and segid H ) )
80         group=( ( resid 107:212 and segid L ) or ( resid 117:213 and segid H ))
81         drop=40.0
82         nstep=40
83         translation=false  {* Translation of the last group is kept fixed. *}
84 end
85 minimize rigid
86         group=( resid 1:106 and segid L )      {*VL: variable, light chain.*}
87         group=( resid 107:212 and segid L )    {*CL: constant, light chain.*}
88         group=( resid 1:116 and segid H )      {*VH: variable, heavy chain.*}
89         group=( resid 117:213 and segid H )    {*CH1:constant, heavy chain.*}
90         drop=40.0
```

```
        nstep=100
        translation=false  {* Translation of the last group is kept fixed. *}
end


                                        {*Turn the crystal symmetry on to  *}
                                        {*carry out the translation search.*}
xrefin
{===>}
   symmetry=(x,y,z)                {*Operators for crystal symmetry P2(1).*}
   symmetry=(-x,y+1/2,-z)
   reduce
end


                                    {*Get ready for the translation search.*}
xrefin
                        {*Set the grid size for the translation search;    *}
                        {*it should be less than 1/3 high-resolution limit.*}
                               {*This is a conservative estimate.  In     *}
                               {*most cases it will be sufficient to double*}
                               {*this estimate.                           *}

   evaluate ( $gridx=2./44. )
   evaluate ( $gridz=2./80. )
   evaluate ( $grid=min($gridx,$gridz) )
   search translation

      packing=true                          {* Packing analysis turned on. *}

      mode=fractional
      xgrid=0.0 $grid 0.5           {*We have to search only in x,z in this*}
      ygrid=0. 0. 0.                {*space group.  In general, we have to *}
      zgrid=0. $grid 0.5            {*specify an asymmetric unit for the   *}
                                    {*translation function.  The basis     *}
                                    {* vectors of the Eucledian normalizer *}
                                    {* of the space group provides an      *}
                                    {* asymmetric unit.                    *}

      nlist=100                     {*List the 100 best grid points;       *}
                                    {*the list is returned in the standard *}
                                    {*output file.                         *}

      output=translation2.3dmatrix            {*Output matrix for plotting.*}
                                              {*This can be verbose for 3d *}
                                              {*translation functions!     *}

   end
end


write coordinates output=translation2.pdb end  {*The translation function. *}
                                               {*Returns the coordinates of*}
                                               {*best solution.            *}

xrefin
   resolution 6. 3.5                    {*Analyze the R-factor distribution*}
   target=residual                      {*of the best solution.            *}
   update
   print r
end

stop
```

This procedure yields a single strong peak for the position of molecule B.

17.5.8 Combined Translation Function to Determine the Relative Position between A and B

We still have to determine the relative z translation between molecules A and B. The following X-PLOR script file accomplishes this. This is also an example of a translation function with a fixed portion (molecule A). The x, z positions of molecules A and B are known up to a 0.5 translation. Therefore, four translation functions have to be carried out.

```
remarks file xtalmr/translation12.inp  -- Combined translation search to
remarks                            find the z-position of the 2nd
remarks                            molecule

{===>} structure @fab2hfl.psf end                    {*Read structure file.*}

{===>} parameter @TOPPAR:parhcsdx.pro end            {*Read parameter file.*}

xrefin
{===>}                                              {*Unit cell for crystal.*}
   a=44.144 b=164.69 c=70.17 alpha=90. beta=108.50 gamma=90.

{===>}
   symmetry=(x,y,z)                 {*Operators for crystal symmetry P2(1).*}
   symmetry=(-x,y+1/2,-z)


   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=30000
   reflection @3r9a_det.fob end                        {*Read reflections.*}

{===>}
   resolution 15.0 4.0                                 {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000


{===>}
   method=fft
   fft
       memory=2000000                  {*FFT method with memory statement.*}
   end

   target=E2E2                      {*Select target for translation search.*}
   mbins=20                                {*Number of bins to compute Es.*}

end

                     {*Get coordinate set from first translation function.*}
{===>}
coor @translation1.pdb

```

```
                          {*Calculate structure factors and store in FPART.*}
xrefin
   update
   print r
   do complex (fpart=fcalc)
end

             {*Get the coordinate set from the second translation function.*}
coor @translation2.pdb
coor copy end              {*Save the coordinate set in the comparison set.*}


            {*The x, z position of the second molecule is known up to a 0.5*}
            {*translation in either x or z or both.  Therefore we have to  *}
            {*run four translation functions to find the solution.         *}

xrefin
   evaluate ( $gridy=2./164. )
   search translation
      mode=fractional
      xgrid=0. 0. 0.
      ygrid=0. $gridy 1.
      zgrid=0. 0. 0.
      nlist=20
      output=translation121.3dmatrix          {*Output matrix for plotting.*}
   end
end

coor swap end
coor copy end
coor fract end
coor translate vector=( 0.5 0. 0. ) end
coor orth end
xrefin
   search translation
      mode=fractional
      xgrid=0. 0. 0.
      ygrid=0. $gridy 1.
      zgrid=0. 0. 0.
      nlist=20
      output=translation122.3dmatrix          {*Output matrix for plotting.*}
   end
end

coor swap end
coor copy end
coor fract end
coor translate vector=( 0.5 0. 0.5 ) end
coor orth end
xrefin
   search translation
      mode=fractional
      xgrid=0. 0. 0.
      ygrid=0. $gridy 1.
      zgrid=0. 0. 0.
      nlist=20
      output=translation123.3dmatrix          {*Output matrix for plotting.*}
   end
end


coor swap end
coor copy end
coor fract end
coor translate vector=( 0. 0. 0.5 ) end
coor orth end
xrefin
   search translation
      mode=fractional
      xgrid=0. 0. 0.
```

```
128       ygrid=0. $gridy 1.
129       zgrid=0. 0. 0.
130       nlist=20
131       output=translation124.3dmatrix          {*Output matrix for plotting.*}
132    end
133 end
134
135 stop
136
```

The first translation function is successful with a correlation coefficient of 0.45. The corresponding Mathematica plot is shown in Fig. 17.5.

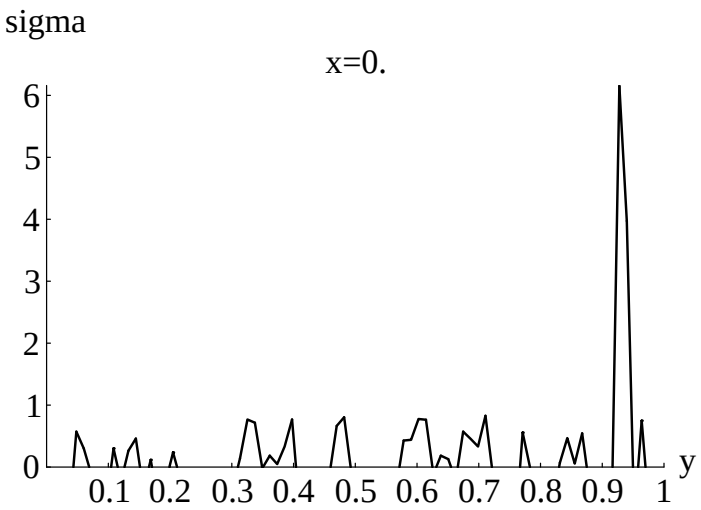

Figure 17.5: Translation function.

17.6 The "Direct" Rotation Function

In this section we describe an alternative strategy to the generalized molecular replacement strategy of Section 17.5. The key idea is to break-up the molecule into rigid domains and to carry out rotation searches with each of the individual domains, followed by translation searches and PC-refinement to find the relative positions of the domains.

Experience shows that many rotation functions including the one described in Section 17.1 fail when the search fragment is too small (Brünger et al. 1991). Furthermore, high-symmetry space groups or poor models often produce fairly weak signals for the rotation function. In this case, the noise produced by the numerical approximations (interpolation or spherical harmonic expansion) used to compute the rotation function can make it impossible to detect the correct solution. These difficulties can be reduced when computing PC (Eq. 17.8) for each sampled orientation Ω from the atomic model placed according to Ω. This "direct" rotation function was in-

strumental for the solution of a GTP analog and the c-H-*ras* P21 Catalytic Domain by Brünger *et al.* (1990b).

The following input file illustrates a direct rotation function for the constant domain of the Fab 26-10 Fab fragment. Note that this direct search strategy can be quite CPU time-consuming.

```
remarks file:  xtalmr/search.inp
remarks "Direct" rotation search using the constant domain only.

{===>} structure @fab2hfl.psf end                    {*Read structure file.*}

{===>} coor @fab2hfl.pdb                               {*Read coordinates.*}

evaluate ($wa=10000.)                {* Use this weight if the only active *}
                                     {* energy term is XREF, otherwise, use*}
                                     {* the weight obtained from the       *}
                                     {* "check.inp" protocol.              *}


xrefin
{===>}                                              {*Unit cell for crystal.*}
   a=44.144 b=164.69 c=70.17 alpha=90. beta=108.50 gamma=90.

{===>}
   symmetry=(x,y,z)                {*Operators for crystal symmetry P2(1).*}
   symmetry=(-x,y+1/2,-z)


   SCATter ( chemical C* )
     2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
     12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
     3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
     6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
     6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
     11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=50000
   reflection @3r9a_det.fob end                        {*Read reflections.*}

{===>}
   resolution 10.0 4.                                  {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000


{===>}
   method=fft
   fft
       memory=1000000                  {*FFT method with memory statement.*}
   end

   wa=$wa
   target=E2E2                {*Specify target used for both PC-refinement*}
                              {*and translation search.                    *}
   mbins=20                       {*Number of bins used for E calculation.*}

   tolerance=0. lookup=false             {*This makes the minimizer happy.*}

                                         {*Expand data to a P1 hemisphere.*}
   expand
```

```
63
64    selection=(( resid 107:212 and segid L ) or ( resid 117:213 and segid H ))
65 end
66
67 flags exclude * include xref end              {*Use only XREF energy terms.*}
68
69 {===>} set display=search.dat end                              {*Output file.*}
70
71 coor copy end
72
73 evaluate ($pcmax=-9999.)
74 evaluate ($pcave=0.)
75 evaluate ($pc2ave=0.)
76 evaluate ($count=0.)
77
78 set message=off end
79 set echo=off end
80
81 evaluate ($t1=0. )                                      {*Theta1 start value.*}
82 evaluate ($t1_int=5.)                                     {*Theta1 interval.*}
83 evaluate ($t1_end=360.01)                                {*Theta1 end value.*}
84 while ($t1 < $t1_end ) loop m1
85   evaluate ($t2=0. )                                    {*Theta2 start value.*}
86   evaluate ($t2_int=5.)                                    {*Theta2 interval.*}
87   evaluate ($t2_end=90.01)                             {*Theta2 ending value.*}
88   while ($t2 < $t2_end ) loop m2
89      evaluate ($t3=0. )                                 {*Theta3 start value.*}
90      evaluate ($t3_int=5.)                                 {*Theta3 interval.*}
91      evaluate ($t3_end=360.01)                            {*Theta3 end value.*}
92      while ($t3 < $t3_end ) loop m3
93         coor swap end
94         coor copy end
95         coor rotate euler=( $t1 $t2 $t3 ) end
96         xrefin
97           update
98           print target
99         end
100        display   $t1[F12.5]  $t2[F12.5]  $t3[F12.5]  $corr[F12.5]
101
102        if ($corr > $pcmax) then
103           evaluate ($pcmax=$corr)
104           evaluate ($t1max=$t1)
105           evaluate ($t2max=$t2)
106           evaluate ($t3max=$t3)
107        end if
108        evaluate ($pcave=$pcave+$corr)
109        evaluate ($pc2ave=$pc2ave+$corr^2)
110        evaluate ($count=$count+1)
111
112        evaluate ($t3=$t3+$t3_int)
113     end loop m3
114   evaluate ($t2=$t2+$t2_int)
115   end loop m2
116 evaluate ($t1=$t1+$t1_int)
117 end loop m1
118
119 close search.dat end
120
121 {====>} set display=search.summary end         {* Output file for summary. *}
122
123 evaluate ($pcave=$pcave/$count)
124 evaluate ($pc2ave=sqrt($pc2ave/$count-$pcave^2))
125
126 display   max. peak  (theta1,       theta2,       theta3,        value):
127 display          $t1max[F12.5]  $t2max[F12.5]  $t3max[F12.5]  $pcmax[F12.5]
128 display   mean of rotation function: $pcave[F12.5]
129 display   standard deviation around mean: $pc2ave[F12.5]
130
131
132       {* To sort the rotation function values we can use system utilities. *}
```

```
133
134       {* UNIX operating systems: *}
135 system  sort -o search.sort +3 -n -r search.dat
136
137       {* VMS operating systems: *}
138 !system sort/key=(position:43,size:14,decimal,desc) search.dat search.sort
139
140
141 stop
```

A summary of the rotation function is written to the file "search.summary". It consists of the maximum value of the rotation function, the mean and the standard deviation around the mean. All rotation function values are written to the file "search.dat". This list needs to be sorted by value. This can be accomplished by operating system services. In the above input file the UNIX sort program is called directly from X-PLOR using the SYSTem statement (cf. Section 2.8). The following shows a few lines of the sorted file "search.sort":

```
     280.00000      35.00000     315.00000       0.07377
     275.00000      35.00000     320.00000       0.06562
     280.00000      35.00000     135.00000       0.06513
     285.00000      35.00000     310.00000       0.05991
     280.00000      35.00000     320.00000       0.05792
     285.00000      35.00000     130.00000       0.05612
     275.00000      35.00000     315.00000       0.05609
     275.00000      35.00000     140.00000       0.05521
     285.00000      35.00000     315.00000       0.05337
     250.00000      30.00000      10.00000       0.04937
     280.00000      35.00000     140.00000       0.04883
     280.00000      30.00000     315.00000       0.04851
     275.00000      35.00000     135.00000       0.04822
```

The two highest peaks are (θ_1=280., θ_2=35., θ_3=315.) and (θ_1=280., θ_2=35., θ_3=135.) which are related by the NCS symmetry of the crystal. They correspond to the orientations of the two Fab fragments in the asymmetric unit of the crystal. An error peak at (θ_1=250., θ_2=30., θ_3=10.) is significantly lower than the two highest peaks. In contrast, the conventional rotation function did not produce any significant solution (Section 17.5.3).

The next step involves finding the orientation of the variable domain of the Fab fragment. One expects that the variation of the interdomain angle between the search model and the crystal structure to be within around 30°. One can therefore restrict the rotation function for the variable domain to a neighborhood around the solution that is provided by the rotation function on the constant domain.

The following input file illustrates a direct rotation function for the variable domain of the Fab 26-10 Fab fragment around the orientation (θ_1=280., θ_2=35., θ_3=315.).

```
1 remarks file:  xtalmr/searchv.inp
2 remarks "Direct" rotation search using the variable domain only
3 remarks  Search around orientation theta1, theta2, theta3= 280, 35, 315.
4
5 {===>} structure @fab2hfl.psf end                    {*Read structure file.*}
6
```

```
{===>} coor @fab2hfl.pdb                                {*Read coordinates.*}

evaluate ($wa=10000.)                {* Use this weight if the only active *}
                                     {* energy term is XREF, otherwise, use*}
                                     {* the weight obtained from the       *}
                                     {* "check.inp" protocol.              *}


xrefin
{===>}                                                {*Unit cell for crystal.*}
   a=44.144 b=164.69 c=70.17 alpha=90. beta=108.50 gamma=90.

{===>}
   symmetry=(x,y,z)                 {*Operators for crystal symmetry P2(1).*}
   symmetry=(-x,y+1/2,-z)


   SCATter ( chemical C* )
     2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
     12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=40000
   reflection @3r9a_det.fob end                         {*Read reflections.*}

{===>}
   resolution 15.0 4.5                                  {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000


{===>}
   method=fft
   fft
       memory=1000000                  {*FFT method with memory statement.*}
   end

   wa=$wa
   target=E2E2                {*Specify target used for both PC-refinement*}
                              {*and translation search.                    *}
   mbins=20                       {*Number of bins used for E calculation.*}

   tolerance=0. lookup=false             {*This makes the minimizer happy.*}

                                         {*Expand data to a P1 hemisphere.*}
   expand

   selection=( ( resid 1:106 and segid L ) or ( resid 1:116 and segid H ) )

end

flags exclude * include xref end             {*Use only XREF energy terms.*}

{===>} set display=searchv.dat end                          {*Output file.*}

coor copy end

evaluate ($pcmax=-9999.)
evaluate ($pcave=0.)
```

```
evaluate ($pc2ave=0.)
evaluate ($count=0.)

set message=off end
set echo=off end

evaluate ($t1=250. )                                    {*Theta1 start value.*}
evaluate ($t1_int=5.0)                                     {*Theta1 interval.*}
evaluate ($t1_end=310.01)                                 {*Theta1 end value.*}
while ($t1 < $t1_end ) loop m1
  evaluate ($t2=5. )                                    {*Theta2 start value.*}
  evaluate ($t2_int=5.0)                                    {*Theta2 interval.*}
  evaluate ($t2_end=65.01)                             {*Theta2 ending value.*}
  while ($t2 < $t2_end ) loop m2
     evaluate ($t3=285. )                               {*Theta3 start value.*}
     evaluate ($t3_int=5.0)                                 {*Theta3 interval.*}
     evaluate ($t3_end=345.01)                             {*Theta3 end value.*}
     while ($t3 < $t3_end ) loop m3
        coor swap end
        coor copy end
        coor rotate euler=( $t1 $t2 $t3 ) end
        xrefin
          update
          print target
        end
        display   $t1[F12.5]  $t2[F12.5]  $t3[F12.5]  $corr[F12.5]

        if ($corr > $pcmax) then
           evaluate ($pcmax=$corr)
           evaluate ($t1max=$t1)
           evaluate ($t2max=$t2)
           evaluate ($t3max=$t3)
        end if
        evaluate ($pcave=$pcave+$corr)
        evaluate ($pc2ave=$pc2ave+$corr^2)
        evaluate ($count=$count+1)

        evaluate ($t3=$t3+$t3_int)
     end loop m3
  evaluate ($t2=$t2+$t2_int)
  end loop m2
evaluate ($t1=$t1+$t1_int)
end loop m1

close searchv.dat end

{====>} set display=searchv.summary end       {* Output file for summary. *}

evaluate ($pcave=$pcave/$count)
evaluate ($pc2ave=sqrt($pc2ave/$count-$pcave^2))

display   max. peak  (theta1,       theta2,       theta3,        value):
display          $t1max[F12.5]  $t2max[F12.5]  $t3max[F12.5]  $pcmax[F12.5]
display   mean of rotation function: $pcave[F12.5]
display   standard deviation around mean: $pc2ave[F12.5]



      {* To sort the rotation function values we can use system utilities. *}

      {* UNIX operating systems: *}
system  sort -o searchv.sort +3 -n -r searchv.dat

      {* VMS operating systems: *}
!system sort/key=(position:43,size:14,decimal,desc) searchv.dat searchv.sort


stop
```

The following shows a few lines of the sorted file “searchv.sort”:

```
     260.00000      40.00000     350.00000       0.08375
     255.00000      40.00000     355.00000       0.06984
     260.00000      40.00000     345.00000       0.06537
     265.00000      40.00000     345.00000       0.06460
     255.00000      40.00000     350.00000       0.05377
     260.00000      35.00000     350.00000       0.05306
     265.00000      35.00000     345.00000       0.05124
     240.00000      25.00000     350.00000       0.05083
     255.00000      45.00000     355.00000       0.05073
     255.00000      40.00000     360.00000       0.05068
     265.00000      35.00000     340.00000       0.04980
     265.00000      40.00000     340.00000       0.04868
```

A clear solution emerges at (θ_1=260., θ_2=40., θ_3=350.).

After determining the orientation of the variable and constant domains one needs to find the relative position between the domains. We start with the relative position as found in the search model and carry out a interdomain translation function in the neighborhood of this position. The interdomain translation function operates in $P1$ which means that the absolute position of the fragments is arbitrary. Prior to the interdomain translation function, the orientations of the domains are optimized by PC-refinement.

```
remarks file tfcross.inp   -- translation search between domains


{===>} structure @fab2hfl.psf end                    {*Read structure file.*}

{===>} parameter @TOPPAR:parhcsdx.pro end            {*Read parameter file.*}

{===>} coor @fab2hfl.pdb                               {*Read coordinates.*}

{===>}          {* Rotate constant domain (orientation from searchc.sort) *}
coor rotate euler=( 280. 35. 315.  )
   selection=(( resid 107:212 and segid L ) or ( resid 117:213 and segid H ))
end

{===>}          {* Rotate variable domain (orientation from searchv.sort) *}
coor rotate euler=( 260. 40. 350.  )
   selection=( ( resid 1:106 and segid L ) or ( resid 1:116 and segid H ) )
end

evaluate ($wa=10000.)                                        {* Arbitrary. *}

xrefin
{===>}                                           {*Unit cell for crystal.*}
   a=44.144 b=164.69 c=70.17 alpha=90. beta=108.50 gamma=90.

{===>}
   symmetry=(x,y,z)                {*Operators for crystal symmetry P2(1).*}
   symmetry=(-x,y+1/2,-z)


   SCATter ( chemical C* )
     2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
     12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
```

```
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=60000
   reflection @3r9a_det.fob end                          {*Read reflections.*}

{===>}
   resolution 15.0 4.0                                   {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * step(fobs - 2.0*sigma))     {*Sigma cutoff.*}
   fwind=0.1=100000


{===>}
   method=fft
   fft
       memory=1000000                  {*FFT method with memory statement.*}
   end

   wa=$wa
   target=E2E2                {*Specify target used for both PC-refinement*}
                              {*and translation search.                    *}
   mbins=20                       {*Number of bins used for E calculation.*}

   tolerance=0. lookup=false             {*This makes the minimizer happy.*}

                                         {*Expand data to a P1 hemisphere.*}
   expand
end

flags exclude * include xref end              {*Use only XREF energy term.*}



xrefin                                                   {* variable domain *}
   selection=( ( resid 1:106 and segid L ) or ( resid 1:116 and segid H ) )
end

{===>}
minimize rigid
   group=( ( resid 1:106 and segid L ) or ( resid 1:116 and segid H ) )
   nstep=20
   drop=40.
   translation=false       {* Translation of the last group is kept fixed. *}
end

minimize rigid
   group=( resid 1:106 and segid L )
   group=( resid 1:116 and segid H )
   nstep=60 drop=40.
   translation=false       {* Translation of the last group is kept fixed. *}
end



xrefin                                                   {* constant domain *}
   sele=( ( resid 107:212 and segid L ) or ( resid 117:213 and segid H ) )
end

{===>}
minimize rigid
   group=( ( resid 107:212 and segid L ) or ( resid 117:213 and segid H ) )
   nstep=20
   drop=10.
   translation=false       {* Translation of the last group is kept fixed. *}
end

minimize rigid
```

```
111    group=( resid 107:212 and segid L )
112    group=( resid 117:213 and segid H )
113    nstep=60 drop=40.
114    translation=false       {* Translation of the last group is kept fixed. *}
115 end
116 
117 
118 xrefin
119    update
120    do complex (fpart=fcalc)
121                                                          {* variable domain *}
122    selection=( ( resid 1:106 and segid L ) or ( resid 1:116 and segid H ) )
123 end
124 
125               {* Safe coordinates; this is necessary since the translation *}
126               {* function applies the best solution to all coordinates     *}
127               {* but we only want to apply it to the variable domain.      *}
128 coor copy end
129                                     {*Get ready for the translation search.*}
130 xrefin
131                         {*Set the grid size for the translation search;    *}
132                         {*it should be less than 1/3 high-resolution limit.*}
133                         {*This is a very conservative estimate.  In many   *}
134                         {*cases it will be sufficient to double this       *}
135                         {*estimate.                                        *}
136 
137    evaluate ( $gridx=1.3 /44. )
138    evaluate ( $gridy=1.3 /160. )
139    evaluate ( $gridz=1.3 /80. )
140 
141    search translation
142 
143       mode=fractional
144       xgrid=-0.1 $gridx 0.1
145       ygrid=-0.1 $gridy 0.1
146       zgrid=-0.1 $gridz 0.1
147 
148 
149 
150 
151       nlist=100                      {*List the 100 best solutions;        *}
152                                      {*the list is returned in the standard*}
153                                      {*output file.                        *}
154 
155    end
156 end
157 
158 coor swap end
159 coor translate vector=( $tfmax_x $tfmax_y $tfmax_z )
160      selection=( ( resid 1:106 and segid L ) or ( resid 1:116 and segid H ) )
161 end
162 
163 xrefin
164    do complex (fpart=0)
165    selection=( all )
166 end
167 
168          {*Do rigid-body minimization of the orientation of both molecules.*}
169 minimize rigid
170    group=( ( resid 1:106 and segid L ) or ( resid 1:116 and segid H ) )
171    group=( ( resid 107:212 and segid L ) or ( resid 117:213 and segid H ) )
172    nstep=20 drop=40.
173    translation=false       {* Translation of the last group is kept fixed. *}
174 end
175                     {*Do rigid-body minimization of the individual domains.*}
176 mini rigid
177    group=( resid 1:106 and segid L )
178    group=( resid 1:116 and segid H )
179    group=( resid 107:212 and segid L )
180    group=( resid 117:213 and segid H )
```

```
   drop=40.0
   nstep=50
   translation=false       {* Translation of the last group is kept fixed. *}
end

write coordinates output=tfcross.pdb end

xrefin
                                       {*Analyze the R-factor distribution*}
   target=residual                     {*of the best solution.            *}
   update
   print r
end

stop
```

The following shows a few lines of the interdomain translation function listing:

```
 Summary of translation search:
   number of grid points=  2275
   mean of T-function=  0.081
   sigma of T-function=  0.007
   maximum of T-function=  0.122
   minimum of T-function=  0.056
 Unit cell coverage for TFmax peak=  0.0526
 List of first    100 highest peaks
    Orthogonal A coordinates      Fractional coordinates
 T=(  -5.528   0.000   3.327) TF=(  -0.100   0.000   0.050) T= 0.1216 P= 0.0526
 T=(  -5.528  -1.372   3.327) TF=(  -0.100  -0.008   0.050) T= 0.1151 P= 0.0526
 T=(  -5.157   0.000   2.218) TF=(  -0.100   0.000   0.033) T= 0.1109 P= 0.0527
 T=(   1.459 -12.352   4.436) TF=(   0.067  -0.075   0.067) T= 0.1089 P= 0.0529
 T=(  -5.899  -1.372   4.436) TF=(  -0.100  -0.008   0.067) T= 0.1085 P= 0.0526
 T=(  -5.899   0.000   4.436) TF=(  -0.100   0.000   0.067) T= 0.1080 P= 0.0526
 T=(  -4.056   0.000   3.327) TF=(  -0.067   0.000   0.050) T= 0.1078 P= 0.0527
 T=(  -4.056  -1.372   3.327) TF=(  -0.067  -0.008   0.050) T= 0.1076 P= 0.0528
```

The highest peak produces a configuration of the two domains that is physically reasonable when inspected by computer graphics.

The resulting coordinates "tfcross.pdb" are used for a conventional translation function similar to "translation1.inp". One obtains a solution that shows similar significance as the one obtained by the generalized strategy in Section 17.5:

```
 Summary of translation search:
   number of grid points=   961
   mean of T-function=  0.135
   sigma of T-function=  0.012
   maximum of T-function=  0.232
   minimum of T-function=  0.104
 Unit cell coverage for TFmax peak=  0.2012
 List of first    100 highest peaks
    Orthogonal A coordinates      Fractional coordinates
 T=(   3.666   0.000   4.436) TF=(   0.117   0.000   0.067) T= 0.2319 P= 0.2012
 T=(   3.295   0.000   5.545) TF=(   0.117   0.000   0.083) T= 0.2150 P= 0.2011
 T=(   4.030   0.000   5.545) TF=(   0.133   0.000   0.083) T= 0.1919 P= 0.2015
```

```
 T=(   2.930   0.000   4.436) TF=(   0.100   0.000   0.067) T= 0.1877 P= 0.2010
 T=(  -0.755   0.000   6.654) TF=(   0.033   0.000   0.100) T= 0.1720 P= 0.2013
 T=(   4.402   0.000   4.436) TF=(   0.133   0.000   0.067) T= 0.1710 P= 0.2015
 T=(  -5.611   0.000  32.163) TF=(   0.117   0.000   0.483) T= 0.1678 P= 0.2011
 T=(  11.017   0.000   6.654) TF=(   0.300   0.000   0.100) T= 0.1672 P= 0.2010
```

The same procedure (rotation function, interdomain translation function, translation function) can be applied to the other Fab fragment in the asymmetric unit. This strategy appears to be robust than the one described in Section 17.5 and it can potentially span a very large range of interdomain angles and positions.

17.7 Rigid-Body Refinement

Now that we know the orientation and position of both molecules, we can carry out rigid-body refinement of all eight domains of the two Fabs. In this particular case, the R value after rigid-body refinement was 41% at 8–3 Å resolution. The following example file also creates a new molecular structure file, which consists of a combination of molecules A and B.

```
remarks file xtalmr/rigid.inp -- Combine molecules and then
remarks                          do rigid-body refinement

{===>} structure @fab2hfl.psf end                    {*Read structure file.*}

{===>} coor @translation1.pdb   {Read coords from 1st translation function.*}


             {*In order to generate the 2nd molecule, we renumber the 1st.*}
vector do ( resid=encode(decode(resid)+1000)) ( all )


{===>} structure @fab2hfl.psf end{*Read structure file for second molecule.*}

{===>} coor @translation2.pdb  {*Read coords from 2nd translation function.*}

evaluate ($wa=10000.)                {* Use this weight if the only active *}
                                     {* energy term is XREF, otherwise, use*}
                                     {* the weight obtained from the       *}
                                     {* "check.inp" protocol.              *}

xrefin
{===>}                                              {*Unit cell for crystal.*}
   a=44.144 b=164.69 c=70.17 alpha=90. beta=108.50 gamma=90.

{===>}
   symmetry=(x,y,z)                 {*Operators for crystal symmetry P2(1).*}
   symmetry=(-x,y+1/2,-z)


   SCATter ( chemical C* )
      2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
```

```
   SCATter ( chemical FE* )
      11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070

{===>}
   nreflections=30000
   reflection @3r9a_det.fob end                          {*Read reflections.*}

{===>}
   resolution 6.0 3.0                                    {*Resolution range.*}

   reduce
   do amplitude ( fobs = fobs * heavy(fobs - 2.0*sigma))    {*Sigma cutoff.*}
   fwind=0.1=100000


{===>}
   method=fft
   fft
       memory=2000000                   {*FFT method with memory statement.*}
   end

   wa=$wa

   tolerance=0. lookup=false              {*This makes the minimizer happy.*}

end


  {*Translate the second molecule according to combined translation search;*}
  {*do the translation in fractional coordinates.                          *}
coor fractionalize end

 {*First apply the x, z translation that was used in job translation12.inp.*}
{===>} coor translate vector=( 0. 0. 0. ) selection=( resid 1:999 ) end

        {*Now apply the translation that was determined by the search in z.*}
{===>} coor translate vector=( 0.  0.957 0. ) selection=( resid 1:999 ) end

coor orthogonalize end


flags exclude * include xref end              {*Use only XREF energy term.*}


         {*Do rigid-body minimization of the orientation of both molecules.*}
minimize rigid
   translation=false  {* Translation of the last group is kept fixed. *}
   group=( resid 1:999 )
   group=( resid 1000:1999 )
   nstep=20 drop=40.
end
                    {*Do rigid-body minimization of the individual domains.*}
mini rigid
        translation=false  {* Translation of the last group is kept fixed. *}
        group=( resid 1:106 and segid L )
        group=( resid 107:212 and segid L )
        group=( resid 1:116 and segid H )
        group=( resid 117:213 and segid H )
        group=( resid 1000:1106 and segid L )
        group=( resid 1107:1212 and segid L )
        group=( resid 1001:1116 and segid H )
        group=( resid 1117:1213 and segid H )
        drop=40.0
        nstep=50
end

xrefin
   update
   print r
end
```

```

write coor output=rigid.pdb end              {*Write combined coordinates.*}
write structure output=rigid.psf end       {*Write combined structure file.*}

stop

```

This completes the structure solution of the 26-10 Fab. Now we are ready for SA-refinement.

17.8 A Packing Function

Sometimes it is useful to compute a packing function. The following example illustrates how to to carry out the packing search:

```
remarks file xtalmr/packing.inp   -- Compute a packing function

{===>} parameter @TOPPAR:parhcsdx.pro end               {*Read parameters.*}

{===>} structure @rigid.psf end                       {*Read structure file.*}

{===>} coor @rigid.pdb                                   {*Read coordinates.*}


xrefin
{===>}                                              {*Unit cell for crystal.*}
   a=44.144 b=164.69 c=70.17 alpha=90. beta=108.50 gamma=90.

{===>}
   symmetry=(x,y,z)                 {*Operators for crystal symmetry P2(1).*}
   symmetry=(-x,y+1/2,-z)

   SCATter ( chemical C* )
       2.31000 20.8439 1.02000 10.2075 1.58860 .568700 .865000 51.6512 .215600
   SCATter ( chemical N* )
      12.2126 .005700 3.13220 9.89330 2.01250 28.9975 1.16630 .582600 -11.529
   SCATter ( chemical O* )
      3.04850 13.2771 2.28680 5.70110 1.54630 .323900 .867000 32.9089 .250800
   SCATter ( chemical S* )
      6.90530 1.46790 5.20340 22.2151 1.43790 .253600 1.58630 56.1720 .866900
   SCATter ( chemical P* )
      6.43450 1.90670 4.17910 27.1570 1.78000 0.52600 1.49080 68.1645 1.11490
   SCATter ( chemical FE* )
     11.1764 4.61470 7.38630 0.30050 3.39480 11.6729 0.07240 38.5566 0.97070


{===>}
   resolution 15.0 4.5                                   {*Resolution range.*}
                                          {*This lower range is sufficient*}
                                          {*for packing analysis.         *}


{===>}
   method=fft
   fft
       memory=2000000                  {*FFT method with memory statement.*}
   end

   target=packing                        {*Use the packing target function.*}
end

                                        {*Get ready for the packing search.*}
xrefin
                                {*Set the grid size for the packing search.*}
   evaluate ( $gridx=2./44. )
```

```
   evaluate ( $gridz=2./80. )
   evaluate ( $grid=min($gridx,$gridz) )
   search translation

      mode=fractional
      xgrid=0.0 $grid 0.5          {*We have to search only in x,z in this*}
      ygrid=0. 0. 0.               {*space group.  In general we have to  *}
      zgrid=0. $grid 0.5           {*specify an asymmetric unit for the   *}
                                   {*translation function.  N.B.: This is *}
                                   {*NOT necessarily identical to an      *}
                                   {*asymmetric unit of the space group!  *}

      nlist=1000                    {*List the 1000 best solutions;       *}
                                    {*the list is returned in the standard*}
                                    {*output file.                        *}

      output=packing.3dmatrix                {*Output matrix for plotting.*}

   end
end

stop
```

It should be noted that for the 26-10 case, the packing function is pretty useless because of the high solvent content and the low crystal symmetry.

17.9 Generation of All Symmetry Mates

It is quite useful to analyze the packing between symmetry-related molecules in the unit cell. While some graphics programs provide a packing-analysis facility, you may find it useful to generate explicitly the symmetry-related molecules within X-PLOR. The following example generates all symmetry mates within the unit cell of the 26-10 Fab example. Note that this example generates positions for backbone atoms only. It would be easy to switch to a different selection of atoms by modifying the delete statement.

```
remarks ../xtalmr/unitcell.inp

{===>}structure @rigid.psf end                    {*Read structure file.*}

{===>}coor @rigid.pdb                               {*Read coordinates.*}

delete                                          {*Keep protein backbone only.*}
  selection=( not (  name ca or name n or name c  ) )
end

xrefin
{===>}                                             {*Unit cell for crystal.*}
   a=44.144 b=164.69 c=70.17 alpha=90. beta=108.50 gamma=90.

end


vector do ( segid="1" ) ( all )

   {*The next two statements would have to be repeated for each space group*}
   {*operator.  Just increase the segment identifier for each new symmetry *}
   {*mate.                                                                 *}
duplicate selection=( segid "1" ) segid="2" end
coor symmetry=(-x,y+1/2,-z) selection=( segid "2" ) end

       {*The next statement translates the molecules into the standard unit*}
```

```
27        {*cell.                                                          *}
28 coor fractionalize end
29 vector do (x=mod(x+10,1)) ( all )
30 vector do (y=mod(y+10,1)) ( all )
31 vector do (z=mod(z+10,1)) ( all )
32 coor orthogonalize end
33
34                      {*A new structure file and coordinate file are written.*}
35 write coordinates output=unitcell.pdb end
36 write structure output=unitcell.psf end
37
38 stop
```

18 Distance Restraints

The NOE statement sets up the database for distance restraints, such as interproton distances. This database is used to evaluate E_{NOE}, which can be combined with other energy terms (see Chapter 4). The resulting hybrid energy function can be used for simulated annealing or minimization, using the methods described in Chapters 9 and 10. (Example protocols for NMR structure determination are described in Chapter 20.)

For further reading, see Brünger (1991b), Brünger et al. (1986), Brünger et al. (1987), Brünger and Karplus (1991), Clore and Gronenborn (1989, 1991), Kaptein et al. (1985), Nilges, Clore, and Gronenborn (1988a, 1988b), Nilges, Gronenborn, and Clore (1989), and Nilsson et al. (1986).

It should be noted that the NOE facility in X-PLOR can be used to restrain any type of distance between selected atoms; e.g., it is possible to restrain the characteristic hydrogen-bonding pattern of secondary structural elements by using the NOE statement.

18.1 Syntax

NOE { <**noe-statement**> } **END** is invoked from the main level of X-PLOR.

<**noe-statement**>:==

ASSIgn <**selection**> <**selection**> <**real**> <**real**> <**real**> adds a distance restraint between the two selected sets of atoms to the NOE database. The interpretation of the real numbers depends on the particular type of restraining function (see Section 18.3). For the POTEntial=BIHA, SQUA, and SOFT options the <real> numbers are d, d_{minus}, and d_{plus} (see also Section 18.4) (default: none).

ASYMptote <***class***> <**real**> is the slope c of the asymptote of the soft-square function (Eq. 18.10) (default: 0).

AVERaging <***class***> **R-6** | **R-3** | **SUM** | **CENTer** specifies the averaging method (Section 18.2) (default: R-6).

BHIG <***class***> <**real**> specifies the height of the potential barriers in the high dimensional restraining function (Eq. 18.15), only for POTEntial=HIGH (default: 0.01).

CEILing=<real> specifies the ceiling value for energy constants; it applies to most restraining functions (Section 18.3) (default: 30).

CLASsification <class-name> partitions the distance restraints database; it applies to all subsequent ASSIgn entries until another CLASS entry is issued (default: NIL).

COUNtviol <class> counts violations in class and stores the smallest violation in the symbol "$RESULT"; action is taken when this statement is issued.

DISTribute <class1> <class2> <real> distributes distance restraints between two classes: the restraint will be put in <class1> if the distance is less than r (the specified real number); it will be put in <class2> if the distance is greater than or equal to r. Action is taken only when this statement is issued.

ENSEmble { <ensemble-statement> } END turns on/off ensemble averaging for NOE distance restraints (default: OFF). The type of averaging is defined by the AVERaging noe-statement. NOTE: ensemble-averaging is NOT compatible with the AVERage=CENTER option. (cf. Section 20.11)

MONOmers <*class*> <integer> specifies the number of monomers for the AVERage=SUM option (Section 18.2) (default: 1).

NCOUnt <*class*> <integer> specifies number of assign statements for the high dimensional restraining function (Eq. 18.15); only even values are allowed (default: 2).

NREStraints=<integer> is a required parameter that specifies the maximum expected number of distance restraints. It has to be greater than or equal to the actual number of restraints. The parameter is used to allocate space dynamically for the restraint list (default: 200).

POTEntial <*class*> BIHArmonic | SQUAre | SOFTsquare | SYMMetry | HIGH | 3DPO specifies the restraining function (Section 18.3) (default: BIHArmonic).

PREDict { <predict-statement> } END can be used to predict interproton distances based on the Cartesian coordinates in the MAIN coordinate set. It lists all distances between the FROM and the TO selections of atoms that are within the range CUTOn ...CUTOff. Distances are R^{-6} averaged over the GROUps that are defined in the topology file for each residue (see Section 3.1.1). Current distance restraint information is overwritten by this statement.

PRINt THREshold=<real> prints distance violations greater than the specified real number. This statement also declares two symbols, $RESULT and $VIOLATIONS, which correspond to the rms deviation of the distances and the number of violations greater than the THREshold value, respectively. The definition of the deviation depends on which

restraining function is specified: in the case of the SQUAre-well and SOFTsquare functions, the deviation refers to the difference between the actual distance and the upper or lower bound if the distance is outside the error range; in the case of the BIHArmonic function, the deviation refers to the difference between the actual distance and the target distance. (A more precise definition is given by Δ in Section 18.3.)

RAVErage <***class****> { <**raverage-statement**> } **END** This statement is used to compute a running-average of each interproton distance. When the running average is turned on, an average distance is accumulated for each interatomic distance defined within the class of noe restraints. This running-average is updated at each time-step in molecular dynamics, and at each step in energy minimization. This facility is described in more detail in Section 18.11.

RESEt erases the current NOE database.

RSWItch <***class****> <**real**> is the switch parameter r_{sw} between the square-well and the asymptote (Eq. 18.10) (default: 10).

SCALe <***class****> <**real**> scales the restraint class (Section 18.3) (default: 1).

SOEXponent <***class****> <**real**> is an exponent; it is for the soft-square function (Eq. 18.10) (default: 2).

SQCOnstant <***class****> <**real**> is an additional scale factor; it is for the square-well or soft-square function (Eqs. 18.8 and 18.10) (default: 20).

SQEXponent <***class****> <**real**> is an exponent; it is for the square-well or soft-square function (Eqs. 18.8 and 18.10) (default: 2).

SQOFfset <***class****> <**real**> applies a negative offset value to all upper bounds d_{plus} of the specified class(es). This offset is applied to the square-well or soft-square functions (Eqs. 18.8 and 18.10) and to the bounds used by distance geometry (see Chapter 19) (default: 0).

TAVErage <***class****> { <**taverage-statement**> } **END** This statement is used to turn on or off time-averaged noe restraints. This facility is described in more detail in Section 18.11.

TEMPerature=<**real**> is the temperature in K; it is only for the biharmonic function (Eq. 18.6) (default: 300).

<**ensemble-statement**>**:==**

ON/OFF turns ON/OFF ensemble averaging. When turned ON only distances involving atoms belonging to identical segid's will be averaged. In case of multimeric proteins (e.g. dimers), specific intermolecular distance restraints can be allowed with the DIMER ensemble-statement (default: OFF).

DIMEr <segid1><segid2> specifies the segid's for allowed intermolecular distances. NOTE: if the dimer statement is used, self-interactions (intramolecular) should ALSO be specified (default: none).

RESEt resets list of allowed intermolecular distances for averaging.

<predict-statement>:==

CUTOff=<real> specifies an upper limit for acceptable interproton distances (default: 10.0 Å).

CUTOn=<real> specifies a lower limit for acceptable interproton distances (default: 0.0 Å).

FROM=<selection> specifies an atom selection (default: none).

TO=<selection> specifies an atom selection (default: none).

<raverage-statement>:==

ON Turns on the running-average.

OFF Turns off the running-average.

EXPOnent=<integer> Specifies the exponent used in distance–averaging (default: 3).

RESEt= CURRent | CONStraint Resets the distance running-averages setting the new initial values to either the current distances (CURRent) or the target distances in the NOE database (CONStraint).

<taverage-statement>:==

ON Turns on time-averaged restraints.

OFF Turns off time-averaged restraints.

TAU=<integer> Specifies the exponential decay constant (in number of time steps) used in the averaging (default: 1000 time steps).

EXPOnent=<integer> Specifies the exponent used in distance–averaging (default: 3).

FORCe=CONServative | NONConservative Specifies whether a conservative or non-conservative force is used in time-averaging (default: CONServative).

RESEt=CURRent | CONStraint Resets the distance time-averages setting the new initial values to either the current distances (CURRent) or the target distances in the NOE database (CONStraint).

18.1.1 Requirements

The molecular structure has to be defined. The atom selections are fragile (Section 2.15).

18.2 Choice of Averaging

For AVERage=R-6, the distance between selected sets of atoms is averaged according to

$$R = (< R_{ij}^{-6} >)^{-1/6} \tag{18.1}$$

where R_{ij} runs through all possible combinations of distance restraints between atoms "$_i$" in set 1 and atoms "$_j$" in set 2.

For AVERage=R-3, the distance between selected sets of atoms is averaged according to

$$R = (< R_{ij}^{-3} >)^{-1/3} \tag{18.2}$$

where R_{ij} runs through all possible combinations of distance restraints between atoms "$_i$" in set 1 and atoms "$_j$" in set 2.

For AVERage=SUM, the distance between selected sets of atoms is computed by adding up single contributions

$$R = (\sum_{ij} R_{ij}^{-6}/n_{mono})^{-1/6} \tag{18.3}$$

where n_{mono} is specified by the MONOmer statement. The scaling by n_{mono} is required, in combination with the SUM averaging option, to scale the distances corresponding to ambiguous peaks in symmetric multimers. The difference between the R-6 option and the SUM option is subtle, and is best illustrated with an example: if two distances are involved in the average, say 3 and 10 Å, the R-6 average will produce an effective distance of 3.37 Å, and the R-6 sum a distance of 2.99 Å, which is probably the desired result for ambiguous NOESY crosspeaks. For this reason, the SUM option should be used rather than the R-6 option to treat ambiguities for observed NOEs.

For AVERage=CENTer, the distance between selected sets of atoms is set to the difference between the geometric centers of the atoms,

$$R = (r_{center}^1 - r_{center}^2) \tag{18.4}$$

18.3 Choice of Restraining Functions

The POTEntial statement allows one to select specific restraining functions for particular classes of distance restraints. By partitioning the distance restraints into particular classes, several restraining functions can be mixed. The total distance restraint energy E_{NOE} is a sum over all classes of distance restraints:

$$E_{NOE} = \sum_{classes} \sum_{restraints} E_{NOE}^{class} \tag{18.5}$$

where the second sum is carried out over all distance restraints of the particular class. Each restraint corresponds to exactly one ASSIgn statement except for the SYMM, 3DPO, and HIGH restraining functions, where each restraint can comprise groups of ASSIgn statements.

18.3.1 Biharmonic Function

For POTEntial=BIHArmonic, a two-sided harmonic function is used:

$$E_{NOE}^{class} = \min(ceil, \frac{SK_bT}{2c_{ij}^2}) \quad (R-d)^2 \tag{18.6}$$

$$c_{ij} = \begin{cases} d_{minus} & R < d \\ d_{plus} & R > d \end{cases} \tag{18.7}$$

where S is the scale factor specified by the SCALe statement, K_b is the Boltzmann constant, T is the temperature specified by the TEMPerature statement, $ceil$ is specified by the CEILing statement, d, d_{plus}, and d_{minus} are specified by the ASSIgn statement:

`assign` < `selection` > < `selection` > d d_{minus} d_{plus}

The distance R is defined in Section 18.2. The deviation Δ is defined as $|R-d|$.

18.3.2 Square-Well Function

For POTEntial=SQUAre-well, a square-well function is used:

$$E_{NOE}^{class} = \min(ceil, SC)\Delta^{exp} \tag{18.8}$$

where Δ is defined as

(18.9)

$$\Delta = \begin{cases} R-(d+d_{plus}-d_{off}) & d+d_{plus}-d_{off} < R \\ 0 & d-d_{minus} < R < d+d_{plus}-d_{off} \\ (d-d_{minus}-R & R < d-d_{minus} \end{cases}$$

S is the scale factor specified by the SCALe statement, C is the scale factor specified by the SQCOnstant statement, and d, d_{plus}, and d_{minus} are specified by the ASSIGn statement:

`assign` < `selection` > < `selection` > d d_{minus} d_{plus}

The exponent exp is specified by the SQEXponent statement and d_{off} is specified by the SQOFfset value. The distance R is defined in Section 18.2.

18.3.3 Soft-Square Function

For POTential=SOFTsquare, the square-well function (Nilges et al. 1988c) is used for distances within a specified "switching" region (r_{sw}, specified by the RSWITch statement), whereas outside this region a "soft" asymptote is used:

(18.10)

$$E_{NOE}^{class} = \min(ceil, SC) \begin{cases} a + \frac{b}{\Delta^{softexp}} + c\Delta & R > d + d_{plus} - d_{off} + r_{sw} \\ \Delta^{exp} & R < d + d_{plus} - d_{off} + r_{sw} \end{cases}$$

where Δ is given by Eq. 18.9 and a and b are determined by the program such that E_{NOE}^{class} is a smooth function at the point $R = d + d_{plus} - d_{off} + r_{sw}$. The slope of the asymptote c is specified by the ASYMptote statement, r_{sw} is specified by the RSWItch statement, and the exponent $softexp$ is specified by the SOEXponent statement. The distance R is defined in Section 18.2.

18.3.4 Symmetry Function

For POTEntial=SYMMetry, a soft-square function (Eq. 18.10) is used to restrain signed distance differences to specified values. The two distances are specified by a pair of ASSIgn statements:

```
assign   < selection >   < selection >   d   d_minus   d_plus
assign   < selection >   < selection >   0   0   0
```

The second distance of a given pair of distances is subtracted from the first; the distance difference can have negative values. The functional form is similar to the soft-square function (Section 18.3.3), except that the switching region is symmetrically arranged around the ideal distance difference (i.e., the program switches to the soft asymptote for $R > d + d_{plus} - d_{off} + r_{sw}$ as well as for $R < d - d_{minus} - r_{sw}$) and Δ is defined as

(18.11)

$$\Delta = \begin{cases} D - (d + d_{plus} - d_{off}) & d + d_{plus} - d_{off} < D \\ 0 & d - d_{minus} < D < d + d_{plus} - d_{off} \\ D - (d - d_{minus}) & D < d - d_{minus} + d_{off} \end{cases}$$

where D denotes the distance difference $R_1 - R_2$ between the two assigned distances. The distances R_1 and R_2 are obtained by various averaging methods as described in Section 18.2.

18.3.5 3D NOE-NOE Function

For POTEntial=3DPO, a restraining function for 3D NOE-NOE experiments is used. The intensity of a 3D NOE-NOE crosspeak, $I_{ijk}(\tau_{m1}, \tau_{m2})$, between spins i, j, and k, is proportional to the product of the individual NOE transfer efficiencies of each mixing time:

$$I_{ijk}(\tau_{m1}, \tau_{m2}) = c[exp(-\mathbf{R}\tau_{m2})]_{ij}[\exp(-\mathbf{R}\tau_{m1})]_{jk} A_{kk}(0), \tag{18.12}$$

where τ_{m1} is the first and τ_{m2} the second mixing time, $\mathbf{R}$ the cross-relaxation matrix, $A_{kk}(0)$ the equilibrium magnetization of spins k, and c a constant.

The 3D NOE-NOE intensity can be approximated by

$$I_{ijk}^{calc} = r_{ij}^{-6} r_{jk}^{-6}. \tag{18.13}$$

where r_{ij} and r_{jk} are the distances between spins i, j and j, k, respectively, and the proportionality constant is set equal to one. The 3D NOE-NOE function is given by

$$E_{NOE}^{class} = S \times \tag{18.14}$$

$$\begin{cases} \{(I_{ijk}^{calc})^{-1/12} - (K(I_{ijk}^{obs} - \Delta_l))^{-1/12}\}^2 & \text{if} \quad I_{ijk}^{calc} < K(I_{ijk}^{obs} - \Delta_l) \\ \{(I_{ijk}^{calc})^{-1/12} - (K(I_{ijk}^{obs} + \Delta_u))^{-1/12}\}^2 & \text{if} \quad I_{ijk}^{calc} > K(I_{ijk}^{obs} + \Delta_u) \\ 0 & \text{else} \end{cases}$$

where S is the energy constant specified by the SCALe statement, I_{ijk}^{obs} is the observed NOE-NOE intensities, Δ_l and Δ_u are error bounds, and K is a scale factor specified by the RSWITch statement. The NOE-NOE connectivity $i \rightarrow j \rightarrow k$ and the intensity and error-bound information are provided by a pair of ASSIgn statements:

`assign`	(spin$_i$)	(spin$_j$)	I_{ijk}^{obs}	Δ_l	Δ_u
`assign`	(spin$_j$)	(spin$_k$)	0	0	0

The first ASSIgn statement specifies spins i and j, along with the observed intensity I_{ijk}^{obs} and the error bounds Δ_l and Δ_u. The second ASSIgn statement specifies spins j and k; the three real numbers following the second selection are irrelevant and can be set to arbitrary values. Note: the 3D NOE-NOE function can be used in combination with R-6 or center averaging. For further details, the reader is referred to Habazettl et al. (1992a, 1992b) and Bernstein et al. (1992).

18.3.6 High dimensional Function

For POTEntial=HIGH, a restraining function for stereospecifically assignable proton pairs is employed (Habazettl et al. 1990). This is useful for resolved

but unassigned peaks involving methylene protons. The restraining function is given by

$$E_{NOE}^{class} = SE_{highdim} \tag{18.15}$$

where S is the energy constant specified by the SCALe statement. Supppose that one observes $2n$ distance restraints involving the two protons a and b of a particular unassigned pair; this means that there are a total of $2n$ ASSIgn statements specified for this particular unassigned pair (NCOUnt has to match $2n$). The $E_{highdim}$ function is then the product of two $2n$-dimensional functions, each corresponding to one of the two possible stereospecific assignments:

$$E_{highdim} = (\sum_{i=1}^{n} E_i^{ass}(r_a^i, r_b^i))(\sum_{i=1}^{n} E_i^{ass}(r_b^i, r_a^i)) \tag{18.16}$$

$$E_i^{ass}(x, y) = C_i(f(x, l_a^i, u_a^i) + f(y, l_b^i, u_b^i) + 1.5F(y - x, L^i, U^i)) \tag{18.17}$$

where r_a^i and r_b^i are the distances from spin i to the unassigned spins a and b, respectively. When the corresponding ASSIgn statements contain multiple atom selections, averaged distances are computed; only center-averaging is allowed in combination with the high dimensional potential. l_a^i and l_b^i are lower bounds for the distances between spin i and spins a and b, respectively. u_a^i and u_b^i are upper bounds for the distances between spin i and spins a and b, respectively. L^i and U^i are the lower bounds and upper bounds for the distance difference $y - x$, respectively. These upper and lower bounds are specified through a group of $2n$ assignment statements:

`assign`	(spin_1)	(spin_a)	l_a^1	u_a^1	L^1
`assign`	(spin_1)	(spin_b)	l_b^1	u_b^1	U^1
`assign`	(spin_2)	(spin_a)	l_a^2	u_a^2	L^2
`assign`	(spin_2)	(spin_b)	l_b^2	u_b^2	U^2
`assign`	(spin_3)	(spin_a)	l_a^3	u_a^3	L^3
`assign`	(spin_3)	(spin_b)	l_b^3	u_b^3	U^3

...

`assign`	(spin_n)	(spin_a)	l_a^n	u_a^n	L^n
`assign`	(spin_n)	(spin_b)	l_b^n	u_b^n	U^n

where the spin_i terms should be understood as atom selections. C_i are constants that are determined internally, such that the barrier height between the two minima of $E_{highdim}$ is given by BHIG. The functional forms of f

and F are similar to the soft-square function (Section 18.3.3) with a linear asymptote, except that the switching region is symmetrically arranged:

$$f(x,l,u) = \begin{cases} 0 & \text{for} \quad l \le x \le u \\ (l-x)^2 & \text{for} \quad x < l \ \text{ and } \ x \ge l - \lim \\ \lim^2 + 2\lim(l-x) & \text{for} \quad x < l - \lim \\ (x-u)^2 & \text{for} \quad x > u \ \text{ and } \ x \le u + \lim \\ \lim^2 + 2\lim(x-u) & \text{for} \quad x > u + \lim \end{cases} \tag{18.18}$$

(18.19)

$$F(\delta,L,U) = \begin{cases} 0 & \text{for} \quad L \le \delta \le U \\ \frac{1}{2}(L-\delta)^2 & \text{for} \quad \delta < L \ \text{ and } \ \delta \ge L - \sqrt{2}\lim \\ \lim^2 + \sqrt{2}\lim(L-\delta) & \text{for} \quad \delta < L - \sqrt{2}\lim \\ \frac{1}{2}(\delta-U)^2 & \text{for} \quad \delta > U \text{ and } \delta \le U + \sqrt{2}\lim \\ \lim^2 + \sqrt{2}\lim(\delta-U) & \text{for} \quad \delta > U + \sqrt{2}\lim \end{cases}$$

where lim is set internally to 0.1.

18.4 Setup of Distance Restraints

Distance restraints consist of one of more ASSIgn statements. It is convenient to include these statements in a file. There are two selection statements that specify the atom pair(s) between which the distance information is included. The first statement selects the first atom (or set of atoms in the case of ambiguous assignments); it has to specify the actual residue number and atom name, e.g.,

```
1 resid 34 and name HN
```

which selects proton HN attached to N of residue 34. In the case of ambiguous assignments, all hydrogens should be selected by using a wildcard specification; e.g.,

```
1 resid 34 and name HB*
```

selects all protons that are bonded to C^{β} (see also the next section). The second statement selects the second atom (or set of atoms).

The file below is an example of a typical distance restraints file obtained from NOE experiments:

```
1 assign (resid  1 and name ha)   (resid  2 and name hn)   2.0 0.0 0.8
2
3 assign (resid  1 and name hg1#)  (resid 31 and name ha)   2.5 0.0 5.5
4 assign (resid  1 and name hg2#)  (resid 31 and name ha)   2.5 0.0 5.5
5
6 assign (resid  1 and name hg1#)  (resid 31 and name hb1)  2.3 0.0 7.3
7 assign (resid  1 and name hg2#)  (resid 31 and name hb1)  2.3 0.0 7.3
8 assign (resid  1 and name hg1#)  (resid 31 and name hb2)  2.3 0.0 7.3
9 assign (resid  1 and name hg2#)  (resid 31 and name hb2)  2.3 0.0 7.3
```

The interpretation of the real numbers is dependent on the particular restraining function used (see Section 18.3).

For distance symmetry restraints, 3D NOE-NOE restraints, and high dimensional restraints, multiple ASSIgn statements can be grouped together. For example, distance symmetry restraints are specified by

```
class sy
  assign (resid 1 and segid A) (resid 30 and segid B) 0.5 0.1 0.1
  assign (resid 1 and segid B) (resid 30 and segid A) 0 0 0
potential sy symm
```

18.5 Pseudoatoms

In X-PLOR, the setup of pseudoatoms is accomplished by the NOE ASSIgn statement (Section 18.1), with multiple protons in either atom selection. For the restraining functions (Section 18.3), X-PLOR computes either an R-6 averaged distance between the involved protons or the distance between the geometric centers of the two specified atom selections (Section 18.2). For distance geometry, X-PLOR automatically applies a pseudoatom correction to the specified distance ranges (see Section 19.2.2).

Pseudoatoms (multiple atom selections) should be used primarily for unresolved NOE cross peaks, like those of methyl groups, prochiral centers, and aromatic rings. In the case of stereospecific assignments, the distances should be exact.

18.6 Incorporation of Other Distance Information

A number of protocols that are described in Chapter 20 incorporate disulfide bonds as distance restraints:

```
assign (resid  3 and name sg) (resid 31 and name sg)  2.02 0.1 0.1
assign (resid  5 and name sg) (resid 20 and name sg)  2.02 0.1 0.1
assign (resid 10 and name sg) (resid 30 and name sg)  2.02 0.1 0.1
```

Hydrogen bonds (e.g., between base pairs in nucleotide structures) can be restrained in a similar manner.

18.7 Dihedral Angle Restraints

The following example shows how to set up torsion angle restraints. Coupling constant measurements provide a means to obtain information about certain dihedral angles, albeit with low precision. The low precision makes it necessary to employ rather large error ranges.

```
restraints
```

```
  dihedral

    reset

    assign (resid 1 and name c) (resid 2 and name ca)
      (resid 2 and name n) (resid 2 and name c)  1 -120.0 40.0 2   {* 9 Hz *}
    assign (resid 2 and name c) (resid 3 and name ca)
      (resid 3 and name n) (resid 3 and name c)  1 -120.0 50.0 2   {* 8 Hz *}

end
```

The four selections of each assign statement specify the particular dihedral angle. The first number after the selections specifies the energy constant (C) in kcal mole^{-1} rad^{-2}, the second number specifies degrees to which the dihedral angle is restrained (ϕ_o), the third number specifies the range around the restrained angle ($\Delta\phi$), and the last number specifies the exponent of the restraining function. (Refer to Section 7.2 for more information.)

18.8 Distance Symmetry Restraints

The symmetry function restrains distance differences between pairs of ASSIgn statements. The distance pairs should be entered as follows:

```
assign (segid a and resid 1 and name ca)                {* First distance.*}
       (segid b and resid 10 and name ca)   0.0 0.1 0.1

                                                       {* Second distance.*}
assign (segid a and resid 10 and name ca)
       (segid b and resid 1 and name ca)   0.0 0.0 0.0
```

This particular example restrains the signed difference between the first and second distance to 0 ± 0.1Å. The values of r, δ_{minus}, and δ_{plus} for the second distance are not used by the program.

The list of symmetry-related intermonomer distance differences for a dimeric protein can be generated automatically by the following script, which produces a file "symrest.tbl":

```
                                  {* Name of the distance restraints file.*}
set display=symrest.tbl end

                                     {* Get number of residues in monomer.*}
vector do (store1 = decode(resid)) (segid a and name ca)
vector show min (store1) (segid a and name ca)
evaluate ($first_residue = $result)
vector show max (store1) (segid a and name ca)
evaluate ($last_residue = $result)

for $id in id (name ca and segid a) loop res1
    vector show element (resid) (id $id)
    evaluate ($resid1 = decode($result))
    evaluate ($resid2 = $last_residue - $resid1 + $first_residue)
    if ($resid2 > $resid1) then
        display ! distance pair $resid1 $resid2
        display assign (resid $resid1 and name ca and segid a)
        display        (resid $resid2 and name ca and segid b) 0 0 0
        display assign (resid $resid1 and name ca and segid b)
        display        (resid $resid2 and name ca and segid a) 0 0 0
    end if
```

```
end loop res1
```

The resulting distance difference list will restrain the distance differences to 0; i.e., it will attempt to produce a perfectly symmetric arrangement of the dimer.

The next example shows how one would actually use the symmetry restraints:

```
noe
  class      symm @symmetry.tbl

  potential  symm symmetry
  scale      symm 1.0
  sqconstant symm 1.0
  sqexponent symm 2
  soexponent symm 1
  rswitch    symm 0.5
  sqoffset   symm 0.0
  asymptote  symm 1.0
end
```

Note that these symmetry distance restraints are more stringent than the non-crystallographic symmetry restraints described in Section 16.1. The distance symmetry implies twofold symmetry whereas the non-crystallographic symmetry restraints simply restrain the monomers to be nearly superimposable without specification of a specific operation between the monomers. The distance symmetry restraints are, however, less stringent than the strict non-crystallographic symmetry (Section 16.2), as the latter requires explicit specification of the symmetry operation. The distance symmetry restraints allow the separation between the monomers to be a self-adjusting parameter. (For further reading, see Nilges and Brünger 1991.)

18.9 3D NOE-NOE Example

The following example shows how to set up the 3D NOE-NOE function for refinement. For further reading, the reader is referred to Habazettl et al. (1992a, 1992b) and Bernstein et al. (1992).

```


noe                                                {* Read NOE restraints.*}
  reset
  nrestraints = 1000                             {* Allocate space for NOEs.*}

  ceiling = 100                 {* Maximum energy constant that is allowed.*}

  class      noenoe

        {* The following is a partial list of the NOE-NOE intensity table. *}
        {* Note that all entries come in pairs; only the first three reals *}
        {* of each pair are important.                                     *}

assign (resid 23 and name hd1#)(resid 29 and name ha1) 0.251 0.23 0.543
assign (resid 29 and name ha1 ) (resid 23 and name hn) 0 0 0

assign (resid 29 and name ha1) (resid 28 and name ha) 0.251 0.178 0.543
```

```
assign (resid 28 and name ha) (resid 29 and name hn)  0 0 0

assign (resid 29 and name ha2) (resid 28 and name ha) 0.501 0.35 1.08
assign (resid 28 and name ha) (resid 29 and name hn) 0 0 0


  averaging  noenoe cent
  potential  noenoe 3dpo
  scale      noenoe 50.                    {* This is the energy constant. *}
  rswitch    noenoe 0.000002801    {* This scales the observed intensities.*}
  ncount     noenoe 2

end
```

18.10 Example for a High Dimensional Restraining Function

The following example shows how to set up the high dimensional restraining function (Eq. 18.15) as described by Habazettl et al. (1990). For one NOE pair per unassigned proton pair, one should specify two ASSIgn statements along with NCOUnt set to 2; for two NOE pairs per unassigned proton pair, one should specify four ASSIgn statements along with NCOUnt set to 4, and so on. In the first pair of assignment statements below involving residue 17, the lower distance bound l_a^1 is set to 2 Å, the upper distance bound u_a^1 is set to 2.9 Å, the lower distance bound l_b^1 is set to 2.8 Å, and the upper distance bound u_b^1 is set to 3.7 Å. The lower distance difference bound L_1 is set to 0.6 Å while the upper distance difference bound U_1 is set to 1.0 Å.

```
  class     2dim
  assign (resid 17 and name hn)  (resid 17 and name hb1)   2.0  2.9  0.6
  assign (resid 17 and name hn)  (resid 17 and name hb2)   2.8  3.7  1.0

  assign (resid 2 and name hn)   (resid 1 and name hg1)    3.2  4.7 -0.3
  assign (resid 2 and name hn)   (resid 1 and name hg2)    3.2  4.7  0.3
  potential 2dim highdim
  average   2dim center
  ncount    2dim 2
  bhig      2dim 0.01

  class     4dim
  assign (resid 11 and name hn)  (resid 11 and name hb1)   3.2  4.7 -1.1
  assign (resid 11 and name hn)  (resid 11 and name hb2)   2.5  3.4 -0.5
  assign (resid 12 and name hn)  (resid 11 and name hb1)   3.0  5.5 -1.5
  assign (resid 12 and name hn)  (resid 11 and name hb2)   2.5  5.1 -0.7
  potential 4dim highdim
  average   4dim center
  ncount    4dim 4
  bhig      4dim 0.01

  class     6dim
  assign (resid 29 and name hn)  (resid 23 and name hd1#)   2.8  4.7 -1.5
  assign (resid 29 and name hn)  (resid 23 and name hd2#)   3.2  6.9  4.1
  assign (resid 29 and name ha1) (resid 23 and name hd1#)   2.1  4.3 -1.3
  assign (resid 29 and name ha1) (resid 23 and name hd2#)   3.0  6.5  4.4
  assign (resid 29 and name ha2) (resid 23 and name hd1#)   2.1  4.0 -1.3
  assign (resid 29 and name ha2) (resid 23 and name hd2#)   3.0  6.5  4.4
  potential 6dim highdim
  average   6dim center
  ncount    6dim 6
```

```
32  bhig      6dim 0.01
```

The high dimensional restraints can be combined with other restraining classes and used in all of the protocols described in section 20.

18.11 Refinement Using Time-Averaged Distance Restraints

NMR-derived structures can be refined with time-averaged NOE distance restraints (Torda et al., 1989, 1990; Pearlman & Kollman, 1991) using the TAVErage statement. In this method the NOE restraint potential is changed so that distance restraints derived from NOE are applied to the time-average of each distance, rather than each instantaneous distance. Thus R in Eqs. 18.6, 18.9, 18.10, 18.11 is replaced by an averaged distance

$$\overline{R}(t) = \left(\frac{1}{\tau_t} \int_0^t e^{-t'/\tau_t} R(t-t')^{-m} \, dt' \right)^{-1/m} \tag{18.20}$$

where τ_t is the characteristic time for the exponential decay (in seconds), and the integral is taken over all time steps since the time-averaging was turned on or reset. The exponential used for distance averaging, m, is set by the EXPOnent statement. The NOE signal arises from dipolar interaction, and hence is a function of R^{-6}; however, for times much less than the correlation time for molecular tumbling, the NOE signal varies as R^{-3}. Hence m should be set to 3 for molecular dynamics simulations in the picosecond time regime. The exponential decay term (e^{-t'/τ_t}) ensures that the rates of change of $\overline{R}(t)$ and E_{NOE} remain approximately equally responsive to the current structure throughout the trajectory. Time-averaging is not possible with the 3D NOE-NOE potential and the high-dimensional potential.

In practice, a slightly different form of the above equation is used to calculate $\overline{R}(t)$; for a discrete number of time points, the equation becomes

$$\overline{R}(t) = \left(\frac{\Delta t}{\tau_t} \sum_{t'=0}^{t} e^{-t'/\tau_t} R(t-t')^{-m} \right)^{-1/m} \tag{18.21}$$

where Δt is the length of one time step. To avoid having to store all distances and re–evaluate the sum at each time point, we use a recursive form of this equation:

$$\overline{R}(t) = \left(\frac{R(t)^{-m}}{\tau} + e^{-1/\tau} \overline{R}(t-\Delta t)^{-m} \right)^{-1/m} \tag{18.22}$$

where $\tau = \tau_t / \Delta t$, and the step size is assumed to be constant. The time-constant for τ is in units of time-steps of the molecular dynamics simulation.

The initial values for $\overline{R}(t)$ can be set to either the current distances, $R(t)$, or to the restraint distances, d, using the TAVErage RESEt statement (CURRent or CONStraint).

The force associated with each NOE restraint is normally taken to be the spatial derivative of the energy term, e.g. for a square well potential,

$$\overline{F}(t) = \nabla E_{NOE} = -\frac{nK_{NOE}}{2}(\overline{R}(t) - d)^{n-1}\nabla\overline{R}(t). \tag{18.23}$$

From Eq. 18.20,

$$\nabla\overline{R}(t) = \frac{1}{\tau}\left[\frac{\overline{R}(t)}{R(t)}\right]^{m+1}\nabla R(t). \tag{18.24}$$

The exact form of $\nabla R(t)$ depends on how many atoms are involved in the NOE restraint, and the choice of averaging. In the simplest case where just two atoms are involved,

$$\nabla R(t) = \frac{\mathbf{R}(t)}{R(t)} \tag{18.25}$$

where $\mathbf{R}(t)$ denotes the vector joining the positions of the two atoms. Thus in the usual case, where $n = 2$ and $m = 3$,

$$\overline{\mathbf{F}}(t) = -\frac{K_{NOE}}{\tau}(\overline{R}(t) - d)\left[\frac{\overline{R}(t)}{R(t)}\right]^{4}\frac{\mathbf{R}(t)}{R(t)} \tag{18.26}$$

Note the fourth–power term with respect to $\overline{R}(t)/R(t)$; this may give rise to occasional large forces. To circumvent this problem, Torda et al. (1989) proposed an alternative force:

$$\overline{\mathbf{F}}(t) = -K_{NOE}(\overline{R}(t) - d)\frac{\mathbf{R}(t)}{R(t)} \tag{18.27}$$

Integrating this force term leads to a time-dependent NOE energy term, hence this force is nonconservative. In X–PLOR the force field can be chosen by setting FORCe to either CONServative (Eq. 18.26) or NONConservative (Eq. 18.27).

X-PLOR can also accumulate running-averages of the distances using the RAVErage statement. The running-average is calculated from

$$< R(t) >= \left(\frac{1}{t}\sum_{0}^{t}\left[R(t-t')\right]^{-3}\delta t'\right)^{-1/3} \tag{18.28}$$

Again, we actually use a recursive form:

$$< R(t) >= \left(\frac{R(t)^{-3} + (N-1) < R(t-\Delta t) >^{-3}}{N}\right)^{-1/3} \tag{18.29}$$

where N is the total number of time (or energy) steps evaluated since RAVEraging was turned on or reset. Note the absence of the the exponential decay term used in Eq. 18.20; this results in all time-steps being weighted equally. This facility is useful to calculate the true average over the course of an entire trajectory.

See Section 20.10 for an example for time- and running-averages.

19 Distance Geometry

This chapter describes how the structure of molecules can be determined using metric matrix distance geometry. For further reading, see Crippen and Havel (1988), Kuszewski, Nilges, and Brünger (1992), and Tarjan (1983).

19.1 Metric Matrix Distance Geometry

19.1.1 Syntax

MMDG { <**mmdg-statement**> } **END** is invoked from the main level of X-PLOR.

<**mmdg-statement**>:==

BACCuracy=<**real**> specifies the value (in Å) that is added to and subtracted from each bond length parameter to give upper and lower bounds on the 1–2 distance (default: 0.01 Å).

EXPOnent <**integer**> specifies the exponent for the DG restraint term for regularization (see Section 19.4).

GROUp=<**selection**> <**real**> selects the atoms that form a rigid group, based on the main coordinate set (see Section 6.1), and the value (in Å) that is added to and subtracted from each interatomic distance calculated from that rigid group. Note that the GROUp statement uses the main coordinate set, in contrast to the REFErence=COORdinate statement and the pseudoatom correction, which use the reference coordinate set. Multiple specifications of GROUps define multiple rigid groups. The group selections need not be disjoint. By default, no rigid groups are defined. The maximum number of groups is currently limited to NATOM, the number of atoms in the molecular structures; the maximum number of all selected atoms in all GROUp statements combined is currently limited to 10*NATOM.

IACCuracy=<**real**> specifies the value (in degrees) that is added to and subtracted from each improper angle parameter defined by atoms i,j,k,l to give upper and lower bounds on the distance between atoms i and l (default: 2°).

METRization <selection> <integer> performs metrization. If this statement is not specified, only bound-smoothing is performed. The statement selects the atoms from which distances can be chosen during the retightening phase of the partial metrization protocol. The integer tells exactly how many of the selected atoms will be used in the retightening phase, which is useful for partial metrization (see Kuszewski, Nilges, and Brünger 1992). If the integer is greater than the number of selected atoms, it is automatically reduced to that number. All other distances are chosen after these and without retightening (although bound smoothing is performed for all atoms).

ORDEred | RANDom are exclusive flags that set the metrization order to either ordered (by internal atom number) or random (default: random).

PACCuracy=<real> specifies the value (in degrees) that is to be added to and subtracted from each dihedral angle parameter defined by atoms i,j,k,l to give upper and lower bounds on the distance between atoms i and l (default: 2°).

READBOunds=<filename> reads smoothed-bound matrices and then performs metrization and embedding.

RECALLbounds recalls smoothed-bound matrices from memory and then performs metrization and embedding. Note that when all atoms are embedded the smoothed-bound matrix will be overwritten with the distances that are picked during the embedding stage. This is done in order to conserve computer memory. If this is not desirable then the WRITEbounds, READbounds statements should be used to write and read the smoothed-bound matrix from disk. However, this problem does not occur when substructures are embedded, since in this case the smoothed-bound matrix will not be overwritten during embedding.

REFErence=PARAmeter | COORdinates uses the parameter database to generate distances for covalent bonds, bond angles, improper angles, and dihedral angles with a single minimum if PARAmeter is specified. If COORdinates is specified, the coordinates from the reference coordinate set (see Section 6.1) are used to obtain the distances. In this case, the accuracy of the distance bounds is specified by BACCuracy, regardless of whether a bond and bond angle define the particular distance. Dihedral angles with multiple minima, nonbonded parameters, and distance and dihedral angle restraints are unaffected by this option (default: PARAmeter).

SCALe <real> specifies the scale factor for the DG restraint term for regularization (see Section 19.4).

SELEction <selection> specifies the atoms that are used in the DG restraint for regularization (see Section 19.4) (default: (NOT ALL)).

SHORtest-path-algorithm=AUTO | FULL | SPARse specifies the algorithm for the shortest path used during metrization and bound smoothing. When AUTO is specified, the program automatically tries to decide which algorithm is more appropriate. If FULL is specified, the Dijkstra algorithm is used. If SPARse is specified, a modified version of the Dijkstra algorithm operates on the (sparse) tree of known distances (default: AUTO).

STOREBounds stores smoothed-bound matrices in memory, and no metrization or embedding is performed. This option saves CPU time when multiple metrizations and embeddings are carried out using the same distance constraints. The option is also required when using the distance geometry restraint term (Section 19.4), which allows the user to regularize coordinates using distance bounds information only.

SUBStructure <selection> specifies the atoms to be embedded (default: (ALL)).

TACCuracy=<real> specifies the value (in degrees) that is to be added to and subtracted from each bond angle parameter to give upper and lower bounds on the 1–3 distance (default: 2°).

VERBose=<logical> switches verbose mode on and off. If verbose mode is on, many status reports are printed (default: off).

WRITEBounds=<filename> writes smoothed-bound matrices to the specified file; no metrization or embedding is performed.

Upon execution of the END statement, the program performs all steps of distance geometry (initialization, metrization, smoothing, and embedding), unless STOREBounds, WRITEBounds, RECALLbounds, or READBOunds is specified.

19.1.2 Requirements

The molecular structure must be defined, the parameters read in, and the distance and dihedral restraints defined by the desired NOE (Chapter 18) and restraints dihedral (Section 7.2) statements before the MMDG statement can be invoked. If rigid groups are to be defined in the MMDG initialize statement, the coordinates after which the groups are modeled must be defined in the main coordinate set. The reference coordinate set has to be defined for the pseudoatom correction and generation of conformational distance bounds if REFErence=COORdinates is specified. The atom selection for the DG energy term is fragile (Section 2.15).

19.1.3 Output

Unless STOREBounds or WRITEBounds is specified, embedded coordinates are written to the main coordinate set, and the following symbols are defined:

$EMBEDDED is a symbol of type string that indicates whether the preceding embedding was successful (string "TRUE") or unsuccessful (string "FALSE").

$DGSCALE is a symbol of type real that holds the distance geometry scaling factor calculated from the preceding embedding.

$MET_GAP is a symbol of type real that holds the average difference between lower and upper bounds that are involved in the metrization stage.

$NON_MET_GAP is a symbol of type real that holds the average difference between lower and upper bounds after smoothing for distances that are not involved in the metrization stage.

19.2 Implementation of Distance Geometry

Many types of structural information (distances, J-coupling data, chemical cross-linking, neutron scattering, predicted secondary structures, etc.) can be conveniently expressed as intra- or intermolecular distances. The distance geometry formalism allows these distances to be assembled and three-dimensional structures consistent with them to be calculated. It will also uncover any inconsistencies in the input data in the process.

19.2.1 Input Distances

The distance geometry routines in X-PLOR begin by translating the bond lengths, bond angles, dihedral angles, improper angles, and van der Waals radii in the current molecular structure into a (sparse) matrix of distance bounds between the bonded atoms, atoms that are bonded to a common third atom, or atoms that are connected to each other through three bonds, as appropriate, using the equations in Crippen and Havel (1988). Experimental constraints can be added by the NOE assign statements (Section 18.1) and the restraints dihedral statements (Section 7.2). These lists of restraints are automatically read, translated into distance constraints, and entered into the bounds matrix.

The data translation is subject to the flag statement (Section 4.5); e.g., to include data from the distance restraint database, the following statement should be issued before invoking the mmdg statement :

```
1 FLAGsINCLude NOE END
```

The definition of upper and lower bounds proceeds in the following order: bond lengths, bond angles, dihedral angles, improper angles, distance and dihedral angle restraints, van der Waals repulsions, and rigid group distances. This order is actually not arbitrary; e.g., the calculation of 1–3 distance

bounds from angles requires that distance bounds among the atoms used to define the parameter (in this case, bond lengths between atoms i and j and atoms j and k, as specified by the parameter statement) be defined previously.

For each bond, the accuracy parameter BACC is added to and subtracted from the equilibrium bond length as specified in the parameter statement (Section 3.2.1) or obtained from the reference coordinate set to give, respectively, the upper and lower bound on the particular 1–2 interatom distance.

Bond angle constraints are obtained from the parameter statement if REFErence=PARAmeter is specified or from the 1–3 distance obtained from the reference coordinate set if REFErence=COORdinates is specified. The accuracy parameter TACC is added to and subtracted from the equilibrium bond angle if REFErence=PARAmeter is specified, and BACC is added to and subtracted from the 1–3 distance if REFErence=COORdinates is specified.

Dihedral angle constraints for single-minimum dihedral angles defined by atoms i,j,k,l are obtained from the parameter statement if REFErence=PARAmeter is specified or from the distance between atoms i and l obtained from the reference coordinate set if REFErence=COORdinates is specified. The accuracy parameter PACC is added to and subtracted from the equilibrium bond angle if REFErence=PARAmeter is specified, and BACC is added to and subtracted from the i,l distance if REFErence=COORdinates is specified. For dihedral angles with multiple minima, the upper and lower bounds correspond to the dihedral angle settings of 180° and 0°. In addition to reading all dihedral angles from the parameter statement database, a generic list comprising all possible dihedral angles in the current molecular structure is generated. The upper and lower bounds for these generic dihedral angles are set to 180° and 0°.

Improper angle constraints defined by atoms i,j,k,l are obtained from the parameter statement if REFErence=PARAmeter is specified or from the distance between atoms i,l obtained from the reference coordinate set if REFErence=COORdinates is specified. The accuracy parameter IACC is added to and subtracted from the equilibrium bond angle if REFErence=PARAmeter is specified, and BACC is added to and subtracted from the i,j distance if REFErence=COORdinates is specified.

For distance restraints (see NOE statement, Section 18.1), the current distance database is read. Upper and lower bounds are set to $d+d_{plus}-d_{off}$, $d-d_{minus}$, where d, d_{plus}, and d_{minus} are specified through the NOE assign statements and d_{off} is specified through the NOE SQOFfset statement (by default d_{off} is zero).

For dihedral angle restraints defined by atoms i,j,k,l (Section 7.2), the database is read as specified by the assign statements and interpreted as upper and lower bounds for the corresponding i,l distance.

To handle the repulsive van der Waals interactions, the nonbonded σ

value (see NONBonded statement in Section 3.2.1) is converted into a van der Waals distance (see Eq. 4.10) and then multiplied by the REPEl parameter as part of the NBONd statement in the parameter statement (see Section 3.2.1). All atom-atom repulsions are included except for 1–3 and 1–2 atom interactions, regardless of the NBXMOd parameter.

The group option calculates the distance between each pair of atoms in the group and enters it into the bounds matrix. It is important that the main coordinate set be well defined for the selected atoms. Multiple GROUp statements define multiple groups. The accuracy parameter specifies a value that will be added to and subtracted from the precise distance as obtained from the coordinates.

19.2.2 Pseudoatoms

In X-PLOR, the setup of pseudoatoms is accomplished by using the NOE assign statement (Section 18.1) with multiple protons in either atom selection. X-PLOR automatically adds a correction to the specified distance bounds, which it derives from the reference coordinate set (see Section 6.1). For example, for a medium NOE from an ALA methyl group to the HN proton of the next residue, one should specify the distance from the geometric center (pseudoatom) of the three methyl hydrogens to the position of the HN proton:

```
1 assign ( resid 1 and HB* ) ( resid 2 and hn ) 3.0 1.8 0.9
```

This assign statement sets the lower bound to 1.2 Å and the upper bound to 3.9 Å. The radius of the methyl group is added automatically by X-PLOR as a correction, since the bounds are applied to all pairs of selected atoms in the distance geometry stage. The specification of the reference coordinate set is mandatory; otherwise a warning message will be printed and no correction applied.

Pseudoatoms (multiple atom selections) should be used primarily for unresolved NOE cross peaks like those of methyl groups, prochiral centers, and aromatic rings. In the case of stereospecific assignments, the distances should be specified explicitly.

An alternative way to replace pseudoatoms is to assign the distance restraint to the heavy atom bound to the protons, in the example above the C^{β} carbon. In this case, one would have to add the radius of the methyl group as the correction, i.e.,

```
1 assign ( resid 1 and CB ) ( resid 2 and hn ) 3.0 1.8 1.8
```

19.2.3 Bound Smoothing

Input distances usually define only a small fraction of the interatomic distances in the system. However, they do imply upper and lower bounds

on the other atoms to which they are connected by means of the triangle inequalities (Crippen and Havel 1988). Determining these implied bounds is equivalent to a well-studied computational problem, finding the shortest path between two points in a directed graph. The shortest-path algorithm used in this implementation is that of Dijkstra (see Tarjan 1983 for a review). It proceeds by calculating the shortest paths from one atom (the root) to all other atoms. The shortest-path routine calculates both upper and lower bounds by keeping track of two halves of the graph, one that holds the upper bounds and one that holds the lower. By solving the shortest-path problem for the upper bounds first, the program can then calculate the lower bounds by continuing the shortest-path routine with the second half of the graph. This process is repeated, with each atom taking its turn as root.

19.2.4 Embedding

Once upper and lower bounds are known for all interatomic distances in the system, a matrix of actual distances chosen between those bounds must be created in order to calculate coordinates. This is done for each pair of atoms in the system with a uniform distribution of random values. Since the initial upper and lower bounds matrices are consistent with all possible conformations of the molecule, interatomic distances chosen independently of each other are, in general, inconsistent with any particular conformation.

The process of calculating coordinates from the resulting actual distance matrix is called embedding. Briefly, the distance matrix is converted to a metric matrix of distances centered around their collective centroid, as described by Crippen and Havel (1988). If the three largest eigenvalues of this metric matrix are all positive, their corresponding eigenvectors contain the x, y, and z coordinates of all the atoms. If the three largest eigenvalues are not positive, the chosen distances are too inconsistent to be embedded. The MMDG statement declares a symbol $EMBEDDED of type string that is set to "TRUE" if the embedding was successful and otherwise to "FALSE".

19.2.5 Scaling

The expected radius of gyration of the embedded coordinates can be calculated from the complete matrix of interatomic distances produced for the embedding. This expected radius is often larger or smaller than the radius of gyration calculated from the embedded coordinates. Therefore, the MMDG statement calculates a scaling factor equal to the ratio of the expected to actual radii of gyration and declares it to be the symbol $DGSCALE. The embedded coordinates should be multiplied by this factor, because it makes subsequent regularization easier.

19.2.6 Metrization

A metrization procedure to improve the consistency of the chosen distances is available. Briefly, the bounds matrix is smoothed for one root, the distance from the root atom to another atom is chosen, and the procedure is repeated, changing roots after the distances from the current root to all other atoms have been chosen. This procedure ensures that later interatom distance choices are consistent with earlier ones. However, it requires considerably more CPU time than the actual embedding because the bounds matrix is frequently resmoothed. It also creates a sampling problem because the order in which distances are chosen has a great impact on which parts of the molecule explore their conformational space completely. If, e.g., the distances are chosen starting from the N terminus of a protein, the molecule's coordinates will be almost completely determined before the distances from the C terminus are chosen. The resulting coordinates show good sampling for the N terminus only, with the C terminus usually lying in an extended loop.

This ordered metrization protocol can be modified to improve the conformational sampling of the coordinates produced. Random metrization picks the root atoms in random order, so that the molecule's conformational freedom is not necessarily used up in just one region.

A modification of these two metrization algorithms in the X-PLOR distance geometry routines gives equally good conformational sampling and reduces the CPU time requirements. In partial metrization, the bounds matrix is resmoothed after choosing distances from only a fraction of the root atoms, after which distances are chosen from the other atoms without resmoothing. This procedure works reliably, even with only four root atoms used in the retightening phase. If very large structures are calculated, the number of root atoms may have to be increased beyond four. The variable $NON_MET_GAP is automatically set by the program to the size of the largest interval in the bounds matrix after (partial) metrization. If $NON_MET_GAP exceeds a few Å, the number of root atoms has to be increased.

19.3 Test for the Correct Enantiomer

Distance geometry cannot distinguish between enantiomers. In fact, distance geometry will more or less randomly produce the correct conformation or its mirror image. Tests have to be carried out in real space to establish the correct enantiomer. Chapter 20 describes a number of ways to carry out these tests with X-PLOR.

19.4 Regularization

X-PLOR can regularize coordinates that are obtained by distance geometry embedding by minimizing the coordinates against an effective energy term (DG) that represents all upper and lower bounds in the distance geometry bounds matrix. The bounds matrix has to be stored in memory using the STOREBounds statement prior to invoking the DG energy term. The atoms that should be restrained have to be defined through the SELEction statement. The DG term is defined as

$$E_{DG} = S \sum_{i<j} \begin{cases} (R_{ij} - d_{ij}^{upper})^e & d_{ij}^{upper} < R_{ij} \\ 0 & d_{ij}^{lower} < R_{ij} < d_{ij}^{upper} \\ (d_{ij}^{lower} - R_{ij})^e & R_{ij} < d_{ij}^{lower} \end{cases} \tag{19.1}$$

where S is the scale factor specified by the SCALe statement, e is the exponent defined by the EXPOnent statement, d_{ij}^{lower} is the upper bounds, d_{ij}^{lower} is the lower bounds generated by the initialization stage of distance geometry, and the sum goes over all atoms i, j that are selected by the selection statement.

The DG energy term is turned on by using the flag statement (see Section 4.5). Normally, all other energy terms should be turned off when using the DG term. However, for defining the bounds matrix initially, the other energy terms should be turned on, because bounds are computed only for the activated energy terms. Then, after storing the bounds matrix, all other energy terms should be turned off and the DG energy term turned on. Energy minimization, molecular dynamics, or any other energy calculation can be performed with the DG energy term.

19.5 CPU and Memory Requirements

The X-PLOR/DG code is highly vectorizable and parallelizable and should perform efficiently on most modern computer architectures.

The complexity of full metrization is $\mathcal{O}(N_{atoms}^3)$, where N_{atoms} is the number of atoms in the (sub) structure. The complexity of bound smoothing alone depends on the number of atoms and on the number of connectivities (known distances) in the molecule. For a molecule consisting of 500 atoms, it takes about 300s on a CONVEX 210 to embed the molecule using partial (4-atom) random metrization.

The X-PLOR/DG module allocates about $N_{atoms} \times N_{atoms}$ single precision words of memory. (Section 20.12 contains more CPU timing results.)

20 NMR Structure Determination

X-PLOR can be used to determine and refine solution NMR structures based on interproton distance estimates, coupling constants measurements, and other information, such as known hydrogen-bonding patterns.

Figure 20.1 presents an overview of distance-based NMR structure determination. There are several alternative methods possible in X-PLOR: full-structure distance geometry, substructure distance geometry, and *ab initio* simulated annealing starting from template structures or random coordinates. The choice of protocol depends on the desired efficiency and sampling of conformational space. Generally, for all well-determined NMR structures, the pathway indicated by black lines should be followed, i.e., substructure embedding and regularization, full-structure SA regularization, and SA refinement. To sample alternative conformations that cannot be sampled with the simple distance geometry substructure approach, the full structure distance geometry protocol with metrization or the *ab initio* simulated annealing protocol starting from a template structure should be used. (For a discussion of sampling characteristics of distance geometry, see Kuszewski, Nilges, and Brünger 1992.) Both options are clearly more time-consuming than substructure distance geometry. For very large structures (larger than 2000 atoms), full-structure distance geometry with metrization is likely to push the memory and CPU limits of most computers. The *ab initio* simulated annealing protocol ("sa.inp") provides an important alternative here. The simulated annealing protocol starting from random coordinates ("random.inp") has been included to provide yet another possibility. It shows that simulated annealing is so powerful that it can convert a random array of atoms into a well defined structure through NOE distance restraints. Structures obtained by this protocol have to be regularized by the "dgsa.inp" protocol.

(For back-calculations and complete matrix relaxation refinement, see Chapter 21.)

Structure determination of extended polypeptides or DNA/RNA double strands is usually underdetermined. In particular, the overall shape or bend of the molecule is a free parameter. Thus, neither distance geometry nor *ab*

initio simulated annealing will produce unique structures. The problem can be avoided by including additional restraints, such as repulsive distance restraints between appropriately chosen phosphate groups in the case of RNA or DNA, or by starting the refinement process from ideal conformations. In the latter case, one should use the "refine.inp" or "refine_gentle.inp" protocols to refine the starting coordinates. It is not necessary to run "generate_template.inp" and the subsequent distance geometry or simulated annealing stages. For example, A- and B-type conformations of DNA or RNA double strands could be obtained from other programs (presently, X-PLOR does not provide a facility to produce ideal DNA conformations). These starting coordinates should be refined and then the convergence of the refined coordinates should be assessed.

If disjoint subunits are present in the molecular structure, distances or covalent bonds need to be defined between them. Otherwise, the relative separation between the subunits is unknown, causing the distance geometry algorithm to place the subunits far apart from each other. This may trigger a message about unknown atomic coordinates if certain atoms have coordinates larger than 9999.9 Å.

The first step of NMR structure determination using X-PLOR involves providing the program with the information it needs about the molecular structure, NOE-derived distance bounds, and coupling-constant-derived dihedral angle restraints. The molecular information of the macromolecule has to be generated using the all-hydrogen force fields "topallhdg.pro", "parallhdg.pro" for proteins and "topallhdg.dna", "parallhdg.dna" for nucleic acids. The generation of the molecular structure should proceed as described in Section 3.7. Note that one has to include all known disulfide bridges in the case of protein structures in the molecular structure generation. Special handling of disulfide bridges and other covalent links occurs in a number of protocols described in this section, and all examples assume that the disulfide bridges have been properly defined. Contrary to earlier versions of X-PLOR, no internal coordinate information is required for any of the NMR protocols.

The "topallhdg.*" force fields for NMR structure determination (Section 3.6) contain improper angles to define the chirality of chiral and prochiral centers. If a coordinate file is used that is not generated by X-PLOR, the local chirality of the coordinates must match the definition used in X-PLOR's parameter sets. The user should check this by computing the energy of the structure with the energy statement (see Section 4.6). The chirality of the chiral and prochiral centers is dependent on the hydrogen naming scheme. An example of how to change the naming convention is provided in the file "topallhdg.convert" in the "toppar" directory (cf. Section 3.6.10). It is clearly imperative that the naming convention used for stereospecific assignments must match the one used by the particular force field.

The setup of NOE-derived distance bounds is described in Chapter 18, and the setup of coupling-constants-derived dihedral angle restraints is de-

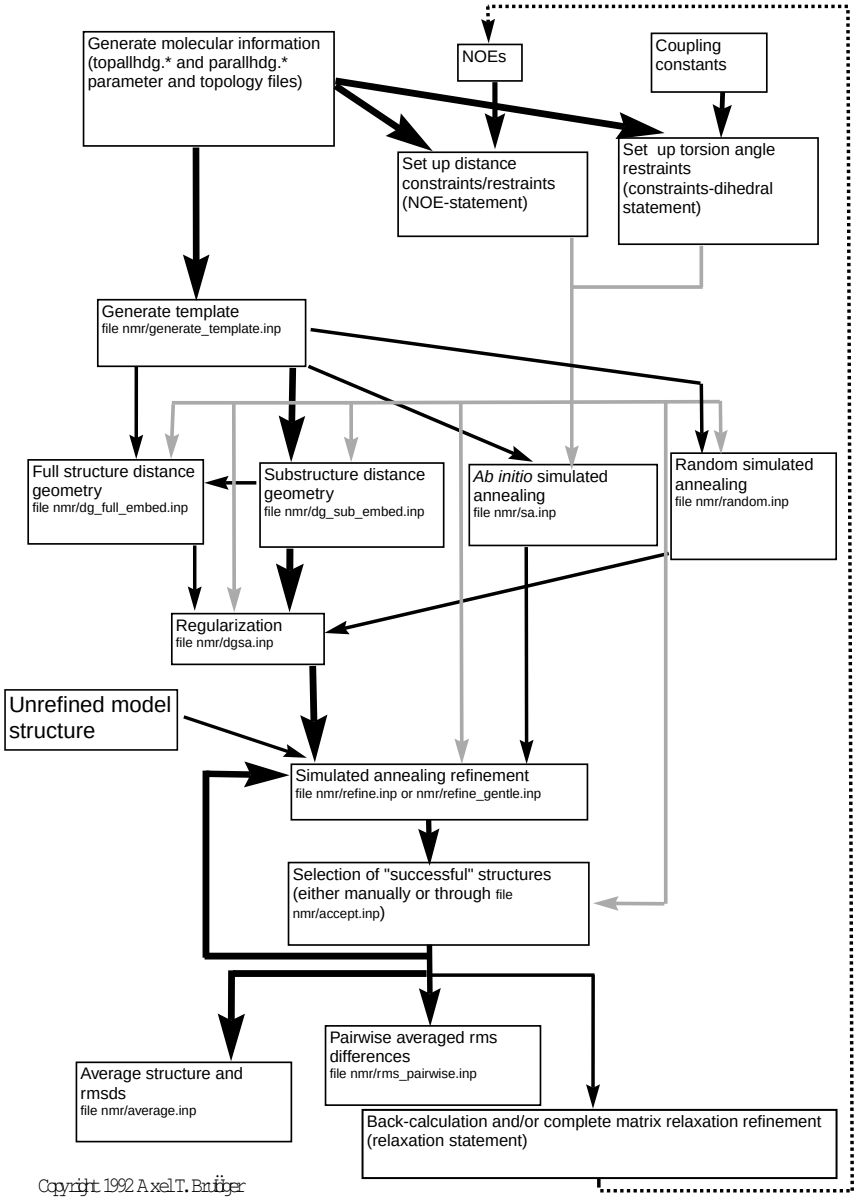

Figure 20.1: Overview of NMR structure determination.

scribed in Section 7.2. For ambiguous or unresolved NOE assignments, distance bounds should be entered that are appropriate for pseudoatoms; i.e., in all examples described in this section, the center-averaging method is used for the NOE restraint function. For example, in the case of a methyl group, the distance is restrained to the geometric center of the methyl hydrogens. This also applies to the distance geometry applications, because the X-PLOR distance geometry routine automatically computes a pseudoatom correction to obtain appropriate distance bounds for each of the methyl group protons

individually.

The final value of the REPEl value (cf. Section 3.2.1) as employed in the simulated annealing and refinement protocols, 0.75, may seem a little small (see also Section 3.6.10). It has been arrived at by inspection of Ramachandran plots of unrestrained polypeptides (Nilges, unpublished results). The allowed region of conformational space is similar to the "generous" region described by Morris et al. (1992). With this value, the final van der Waals energy should be very small. Larger values of REPEl up to 0.8 are also acceptable. The general effect of increasing REPEL will be a reduction of the rms distribution of an ensemble of coordinates, and an increase in the final potential energy.

20.1 Template Structure

The next step involves generation of a template coordinate set. This is required in all cases, except for the random simulated annealing protocol. The template coordinate set can be any conformation of the macromolecule with good local geometry and no nonbonded contacts. It can be generated by using most molecular modeling graphics programs or, preferably, by using the X-PLOR protocol described below. The purpose of the template coordinate set is to provide distance geometry information about the local geometry of the macromolecule, to apply appropriate pseudoatom corrections, and to provide a starting point for the *ab initio* simulated annealing protocol. Furthermore, template coordinates can be used to define large rigid groups in distance geometry, e.g., if certain portions of the protein structure are known by other means or have been determined by other methods.

The protocol below automatically generates a template coordinate set. It initially places the atoms of the macromolecule along the x-axis, with y and z set to random numbers. The coordinates are then regularized using simulated annealing. The protocol is completely general, and it has been tested for both proteins and nucleic acids.

Disulfide bonds and other covalent links between sequentially distant residues may have to be removed for successful completion of the template generation. Generally, when too many covalent links are present, the structure may get entangled in a knot which will result in poor local geometry. Some experimentation may be required to find out if certain covalent links have to be removed; the goal is to obtain an energy below 1000 kcal mole^{-1} for the final step of minimization. This protocol has been successfully used with up to three disulfide bridges. However, in the case of bovine pancreatic trypsin inhibitor, removal of the disulfides was required.

```
1 remarks  file  nmr/generate_template.inp
2 remarks  Generates a "template" coordinate set.  This produces
3 remarks  an arbitrary extended conformation with ideal geometry.
4 remarks  Author: Axel T. Brunger
5
```

```
{====>}
structure @g_protein.psf end                          {*Read structure file.*}


parameter
{====>}
   @TOPPAR:parallhdg.pro                                  {*Read parameters.*}
end


topology
   presidue NDIS
      delete bond 1SG 2SG
      delete angle 1CB 1SG 2SG
      delete angle 1SG 2SG 2CB
   end
end

{====>}
                {*If protein contains S-S bridges, appropriately modify and*}
                {*then uncomment the following lines.                       *}
!patch ndis reference=1=( residue 5  ) reference=2=( residue 55 ) end
!patch ndis reference=1=( residue 14 ) reference=2=( residue 38 ) end
!patch ndis reference=1=( residue 30 ) reference=2=( residue 51 ) end



vector ident (x) ( all )
vector do (x=x/10.) ( all )
vector do (y=random(0.5) ) ( all )
vector do (z=random(0.5) ) ( all )

vector do (fbeta=50) (all)                {*Friction coefficient, in 1/ps.*}
vector do (mass=100) (all)                        {*Heavy masses, in amus.*}

parameter
   nbonds
      cutnb=5.5 rcon=20. nbxmod=-2 repel=0.9  wmin=0.1 tolerance=1.
      rexp=2 irexp=2 inhibit=0.25
   end
end

flags exclude * include bond angle vdw end

minimize powell nstep=50  nprint=10 end

flags include impr end

minimize powell nstep=50 nprint=10 end

dynamics  verlet
   nstep=50  timestep=0.001 iasvel=maxwell  firsttemp= 300.
   tcoupling = true  tbath = 300.   nprint=50  iprfrq=0
end

parameter
   nbonds
      rcon=2. nbxmod=-3 repel=0.75
   end
end

minimize powell nstep=100 nprint=25 end

dynamics  verlet
   nstep=500  timestep=0.005 iasvel=maxwell  firsttemp= 300.
   tcoupling = true  tbath = 300.   nprint=100  iprfrq=0
end

flags exclude vdw elec end
vector do (mass=1.) ( name h* )
```

```
76 hbuild selection=( name h* ) phistep=360 end
77 flags include vdw elec end
78
79 minimize powell nstep=200 nprint=50 end
80                                                    {*Write coordinates.*}
81 remarks produced by nmr/generate_template.inp
82 write coordinates output=generate_template.pdb end
83
84 stop
```

20.2 Options: Distance Geometry, *Ab Initio* SA, or Random SA

The user now has the choice of carrying out either distance geometry, *ab initio* simulated annealing starting from the template coordinates, or simulated annealing starting from random coordinates. The substructure distance geometry protocol, in combination with simulated annealing regularization, is usually more efficient than the pure simulated annealing protcols for well-determined systems. However, if there is any doubt about the uniqueness of the solution, *ab initio* simulated annealing or full-structure distance geometry should also be tried.

20.3 Distance Geometry

The example input file below shows how to produce a family of embedded substructures using distance geometry. The substructures are regularized after embedding using minimization against the DG energy term. The substructure selection depends on the type of macromolecule. The example file has been tested for proteins and nucleic acids.

Disulfide bonds and other covalent links should be treated in the same fashion as in the prior template generation. The removed covalent links are reintroduced as distance restraints.

It is important for the subsequent regularization protocol that each residue of the macromolecule (as defined by atom tags, see Section 2.15) contain at least three atoms. Otherwise, the template fitting to generate the missing atoms will be ill-behaved. The filenames of the family of embedded substructures are "dg_sub_embed_1.pdb" through "dg_sub_embed_10.pdb".

```
1  remarks file nmr/dg_sub_embed.inp --
2  remarks           Bound smoothing, (sub)structure embedding,
3  remarks           and regularization to produce a family of
4  remarks           DG structures.
5  remarks
6  remarks Author: Axel T. Brunger
7
8  {====>}
9  structure @g_protein.psf end                       {*Read structure file.*}
10
11
12 parameter
```

```
{====>}
   @TOPPAR:parallhdg.pro                                {*Read parameters.*}
   nbonds
      repel = 0.8                    {*This scales the van der Waals radii.*}
   end
end

topology
   presidue NDIS
      delete bond 1SG 2SG
      delete angle 1CB 1SG 2SG
      delete angle 1SG 2SG 2CB
   end
end

{===>}
               {*If protein contains S-S bridges, appropriately modify and *}
               {*then uncomment the following lines. The S-S covalent bonds*}
               {*will be deleted and reintroduced as fake NOE distances.   *}
!patch ndis reference=1=( residue 5  ) reference=2=( residue 55 ) end
!patch ndis reference=1=( residue 14 ) reference=2=( residue 38 ) end
!patch ndis reference=1=( residue 30 ) reference=2=( residue 51 ) end


noe
{====>}
   nres=3000                           {*Approximate number greater than the*}
                                       {*actual number of NOEs.             *}
   class = all

{====>}
   @g_protein_noe.tbl                        {*Read NOE distance ranges.   *}
                                             {*Note that no other settings *}
                                             {*are important for embedding.*}
end


{====>}
@g_protein_dihe.tbl                     {*Read dihedral angle restraints.*}


{====>}
               {*If protein contains S-S bridges, appropriately modify and *}
               {*then uncomment the following lines.                       *}
!noe
!   assign (resid  5 and name sg) (resid 55 and name sg)  2.02 0.1 0.1
!   assign (resid  14 and name sg) (resid 38 and name sg)  2.02 0.1 0.1
!   assign (resid 30 and name sg) (resid 51 and name sg)  2.02 0.1 0.1
!end


            {*Read template for pseudoatom correction and for target values*}
            {*for conformational constraints (bonds, angles, etc.).        *}
{===>} coor disp=refe @generate_template.pdb


{===>}
                                {*Store (sub)structure selection in store1.*}
         {*The following substructure selection is typical for a protein;  *}
         {*for nucleic acids try name p or name c3' or name c5' or name c1'*}
         {*or name n9 or name n1 or name c2 or name c4 or name n3.         *}

vector ident ( store1 ) (name ca or name ha or name n or name hn
        or name c or name cb* or name cg*)

                                     {*Energy flags: both NOEs and dihedral*}
                                     {*angle restraints are included.      *}

flags exclude * include bond angle dihedral improper vdw noe cdih end
```

```
83
84 mmdg                                                    {*Create bounds matrix.*}
85    reference=coordinates
86    storebounds                                          {*Store bounds matrix.*}
87 end
88
89                                        {*Include DG term for regularization.*}
90 flags exclude * include dg end
91 constraints interaction=( recall1 ) ( recall1 ) end
92
93 evaluate ($count = 0)
94
95                {*The following loop produces a family of 10 substructures.*}
96 {====>}
97 while ($count < 10 ) loop main
98
99    evaluate ($count=$count+1)
100   evaluate ($embedded = false)
101
102                                       {*Loop until embedding is successful;*}
103                                       {*normally the success rate is high, *}
104                                       {*and it will work during the first  *}
105                                       {*pass.                              *}
106   while ($embedded = false) loop embed
107      mmdg
108         recallbounds                                    {*Get bounds matrix.*}
109
110         substructure=( recall1 )
111
112         selection=( recall1 )                    {*Specify parameters    *}
113         scale=100. exponent=2                    {*for DG-regularization.*}
114      end
115   end loop embed
116
117   vector do (x = x * $dgscale) (known)           {*Scale the structure;  *}
118   vector do (y = y * $dgscale) (known)           {*the symbol $DGSCALE is*}
119   vector do (z = z * $dgscale) (known)           {*defined by MMDG.      *}
120
121   minimize powell                                       {*Regularization.*}
122      nstep=100 drop=10. nprint=25
123   end
124
125 {====>}          {*Uncomment the following lines if a test for the correct *}
126                  {*enantiomer is desired based on improper energy.         *}
127                  {*For this test to be successful, the substructures must  *}
128                  {*contain chiral centers (e.g., CA) and all their ligands.*}
129 !flags exclude dg include impr end
130 !energy end
131 !evaluate ($old_e=$impr)
132 !vector do (x=-x) ( known )
133 !energy end
134 !if ($impr > $old_e) then
135 !  vector do (x=-x) ( known )
136 !end if
137 !flags exclude impr include dg end
138
139 {====>}          {* Uncomment the following lines if a test for the correct*}
140                  {* enantiomer is desired based on an rms difference from a*}
141                  {* reference structure.                                   *}
142 !coor disp=comp @reference.pdb                {*Read reference structure.*}
143 !coor fit sele=( known ) end
144 !coor rms sele=( known ) end
145 !evaluate ($old_rms=$result)
146 !vector do (x=-x) ( known )
147 !coor fit sele=( known ) end
148 !coor rms sele=( known ) end
149 !if ($result > $old_rms) then
150 !  vector do (x=-x) ( known )
151 !end if
152
```

```
153    remarks  produced by nmr/dg_sub_embed.inp
154
155 {====>}                  {*Name(s) of the family of embedded substructures.*}
156    evaluate ($filename="dg_sub_embed_"+encode($count)+".pdb")
157
158    write coordinates output =$filename end
159
160 end loop main
161
162 stop
```

20.3.1 Test for the Correct Enantiomer

Distance geometry cannot distinguish between enantiomers. The test for the correct enantiomer can be carried out either at the substructure level or after the coordinates of all atoms have been generated. The safest procedure is to generate all atomic coordinates for both enantiomers and then to refine both enantiomers. The one with the lower refined energy will almost certainly be the correct enantiomer (see file "dgsa.inp" below). This test is somewhat time-consuming, because one has to carry out a large number of refinement steps for the incorrect enantiomer.

Two other alternatives will work for well defined systems and appropriately defined substructures but may fail for less well defined systems. The first involves testing both mirror images of a particular substructure according to their improper energy. The enantiomer with the lower improper energy is likely to be the correct one. This test will work only when chiral centers are part of the substructure selection; the central atom as well as all of its ligand have to be included. The test also assumes that the improper angles in the parameter and topology file have been well defined. This enantiomer testing method is included in the file "dg_sub_embed.inp" as a comment; to activate it, the user has to uncomment the appropriate lines.

The second alternative involves testing both mirror images of a particular substructure according to its least-squares fit to a known reference coordinate set. The enantiomer with the lower rms difference is very likely to be the correct one. Again, this enantiomer testing method is included in the file "dg_sub_embed.inp" as a comment; to activate it, the user has to uncomment the appropriate lines. This feature will not work in the case of extremely underdetermined systems that exhibit very large rms differences between possible conformations.

20.3.2 Options: Full-Structure Embedding or SA

To obtain coordinates for all atoms, the user has a choice between proceeding directly to the "dgsa.inp" protocol, which generates the missing atoms and then regularizes the coordinates, or embedding all coordinates, using the substructures as additional constraints for the distance geometry program. In the latter case, the coordinates require further regularization with the "dgsa.inp" protocol.

20.3.3 SA-Regularization of DG-Structures

Embedded distance geometry (substructure) coordinates require extensive regularization. The "dgsa.inp" protocol below uses template fitting followed by simulated annealing to regularize the coordinates. The protocol is close in spirit to the one published by Nilges, Clore, and Gronenborn (1988b). However, many improvements have been added to the original protocol, and it appears that the present protocol is more efficient and robust. The example file is general and should work for all types of macromolecules; it has been tested for proteins and nucleic acids. No change is required when using this protocol for nucleic acids compared to proteins. The starting coordinates have to be defined for at least three atoms in each residue (see the definition of atom tags in Section 2.15).

Disulfide bonds and other covalent links are treated as "real" bonds in this protocol, in contrast to template generation.

Important parameters that the user might want to change are the simulated annealing starting temperature, the duration of the high-temperature stage, and the duration of the cooling stage. The example file worked well for proteins and nucleic acids with up to about 2300 atoms. An increase in the duration of the high-temperature and cooling stage is required if the initial distance geometry (sub)structure is far from the correct solution. There is an abort condition that is triggered when the temperature during molecular dynamics rises uncontrollably. This condition requires manual intervention, which involves decreasing the time step of the molecular dynamics portion of the slow-cooling stage of the protocol.

The filenames of the family of regularized coordinates are "dgsa_1.pdb" through "dgsa_10.pdb". The coordinate files contain useful information about the energies, restraint-satisfaction, and violations.

```
remarks file nmr/dgsa.inp  -- Simulated annealing regularization
remarks                       and refinement for embedded distance
remarks                       geometry structures or substructures.
remarks  Authors: Michael Nilges, John Kuszewski, and Axel T. Brunger


{====>}
evaluate ($init_t = 2000)            {*Initial annealing temperature, in K.*}
{====>}
evaluate ($high_steps = 1000 )        {*Total number of steps at high temp.*}
{====>}
evaluate ($cool_steps = 1000 )         {*Total number of steps for cooling.*}

parameter                                        {*Read the parameter file.*}
{====>}
    @TOPPAR:parallhdg.pro
end

{====>}
structure @g_protein.psf end                     {*Read the structure file.*}

noe
{====>}
   nres=3000             {*Estimate greater than the actual number of NOEs.*}
   class all
{====>}
```

```
   @g_protein_noe.tbl                          {*Read NOE distance ranges.*}
end

{====>}
@g_protein_dihe.tbl                       {*Read dihedral angle restraints.*}


vector do (fbeta=10) (all) {*Friction coefficient for MD heatbath, in 1/ps.*}
vector do (mass=100) (all)              {*Uniform heavy masses to speed MD.*}

noe                              {*Parameters for NOE effective energy term.*}
   ceiling=1000
   averaging  * cent
   potential  * square
   sqconstant * 1.
   sqexponent * 2
   scale      * 50.           {*Constant NOE scale throughout the protocol.*}
end

parameter                         {*Parameters for the repulsive energy term.*}
   nbonds
      repel=0.5                    {*Initial value for repel--modified later.*}
      rexp=2 irexp=2 rcon=1.
      nbxmod=-2                   {*Initial value for nbxmod--modified later.*}
      wmin=0.01
      cutnb=4.5 ctonnb=2.99 ctofnb=3.
      tolerance=0.5
   end
end

restraints dihedral
      scale=5.                           {*Initial weight--modified later.*}
end

{====>}
evaluate ($end_count=10)          {*Loop through a family of 10 structures.*}

evaluate ($count = 0)
while ($count < $end_count ) loop main

   evaluate ($count=$count+1)

{====>}                              {*Filename(s) for embedded coordinates.*}
   evaluate ($filename="dg_sub_embed_"+encode($count)+".pdb")

                              {*Test for the correct enantiomer;            *}
{====>}                       {*if you want to bypass this test because the *}
                              {*substructures were tested previously, simply*}
                              {*remove the -1 from the next statement.      *}
for $image in ( 1 -1 ) loop imag
      coor initialize end
      coor @@$filename
      vector do (x=x * $image) ( known )
      vector identity (store1) (not known)        {*Set store1 to unknowns.*}

     {* ================== Create local ideal geometry by template fitting;*}
     {*                    this also takes care of unknown atoms.          *}

      set message=off echo=off end
      coor copy end          {*Store current coordinates in comparison set.*}

{====>}                {*The user has to supply a template coordinate set.*}
      coor @@generate_template.pdb

      for $id in id ( tag ) loop fit              {*Loop over residue tags.*}

         coordinates                {*LSQ fitting using known coordinates.*}
            fit select = ( byresidue (id $id) and not store1 )
         end
      {*Store fitted template coordinates for this residue.*}
```

```
97          coor copy selection=( byresidue (id $id) ) end
98
99        end loop fit
100
101       coor swap end
102       set message=on echo=on end
103
104     {* ============================= Minimization of bonds, VDWs, and NOEs.*}
105       restraints dihedral   scale=5.   end
106       parameter  nbonds   nbxmod=-2 repel=0.5   end  end
107       flags exclude * include bond vdw noe cdih end
108       constraints interaction (all) (all) weights * 1.  vdw 20. end end
109
110       minimize powell nstep=100 nprint=10 end
111
112      {* ================================================ Include angles. *}
113
114       flags include angl end
115
116       minimize powell nstep=100 nprint=10 end
117
118      {* ============ Dynamics, slowly introducing chirality and planarity. *}
119       flags include impr end
120
121       evaluate ($nstep1 = int($high_steps/8))
122       evaluate ($nstep2 = int($high_steps/2))
123
124       constraints inter (all) (all) weights * 0.1 impr 0.05 vdw 20. end end
125       dynamics  verlet
126          nstep=$nstep1 time=0.003 iasvel=maxwell firstt=$init_t
127          tcoup=true tbath=$init_t nprint=100 iprfrq=0
128       end
129       constraints inter (all) (all) weights * 0.2 impr 0.1  vdw 20. end end
130       dynamics  verlet
131          nstep=$nstep1 time=0.003 iasvel=current firstt=$init_t
132          tcoup=true tbath=$init_t nprint=100 iprfrq=0
133       end
134
135       parameter  nbonds repel=0.9   end  end
136       constraints inter (all) (all) weights * 0.2 impr 0.2 vdw 0.01 end end
137       dynamics  verlet
138          nstep=$nstep1 time=0.003 iasvel=current firstt=$init_t
139          tcoup=true tbath=$init_t nprint=100 iprfrq=0
140       end
141
142       parameter nbonds nbxmod=-3  end  end
143       constraints inter (all) (all) weights * 0.4 impr 0.4 vdw 0.003 end end
144       dynamics  verlet
145          nstep=$nstep2 time=0.003 iasvel=current firstt=$init_t
146          tcoup=true tbath=$init_t nprint=100 iprfrq=0
147       end
148
149       constraints inter (all) (all) weights * 1.0 impr 1.0 vdw 0.003 end end
150       dynamics  verlet
151          nstep=$nstep1 time=0.003 iasvel=current  firstt=$init_t
152          tcoup=true tbath=$init_t nprint=100 iprfrq=0
153       end
154
155       if ($image = 1) then
156          vector do (store7=x) ( all )        {*Store first image in stores.*}
157          vector do (store8=y) ( all )
158          vector do (store9=z) ( all )
159          vector do (store4=vx) ( all )
160          vector do (store5=vy) ( all )
161          vector do (store6=vz) ( all )
162       end if
163
164     end loop imag
165
166     {* ================== Establish the correct handedness of the structure.*}
```

```
167
168    energy end
169    evaluate ($e_minus=$ener)
170    coor copy end
171    vector do (x=store7) ( all )
172    vector do (y=store8) ( all )
173    vector do (z=store9) ( all )
174    energy end
175    evaluate ($e_plus=$ener)
176    if ( $e_plus > $e_minus ) then
177       evaluate ($hand=-1 )
178       coor swap end
179    else
180       evaluate ($hand= 1 )
181       vector do (vx=store4) ( all )
182       vector do (vy=store5) ( all )
183       vector do (vz=store6) ( all )
184    end if
185
186
187       {* ============================== Increase VDW interaction and cool.*}
188
189    restraints dihedral   scale=200.   end
190
191    evaluate ($final_t = 100)     { K }
192    evaluate ($tempstep = 50)     { K }
193
194    evaluate ($ncycle = ($init_t-$final_t)/$tempstep)
195    evaluate ($nstep = int($cool_steps/$ncycle))
196
197    evaluate ($ini_rad  = 0.9)        evaluate ($fin_rad  = 0.75)
198    evaluate ($ini_con=  0.003)       evaluate ($fin_con=  4.0)
199
200    evaluate ($bath  = $init_t)
201    evaluate ($k_vdw = $ini_con)
202    evaluate ($k_vdwfact = ($fin_con/$ini_con)^(1/$ncycle))
203    evaluate ($radius=    $ini_rad)
204    evaluate ($radfact = ($fin_rad/$ini_rad)^(1/$ncycle))
205
206    evaluate ($i_cool = 0)
207    while ($i_cool < $ncycle) loop cool
208       evaluate ($i_cool=$i_cool+1)
209
210       evaluate ($bath  = $bath  - $tempstep)
211       evaluate ($k_vdw=min($fin_con,$k_vdw*$k_vdwfact))
212       evaluate ($radius=max($fin_rad,$radius*$radfact))
213
214       parameter  nbonds repel=$radius   end end
215       constraints interaction (all) (all) weights * 1. vdw $k_vdw end end
216
217       dynamics  verlet
218          nstep=$nstep time=0.005 iasvel=current firstt=$bath
219          tcoup=true tbath=$bath nprint=$nstep iprfrq=0
220       end
221
222
223 {====>}                                                 {*Abort condition.*}
224       evaluate ($critical=$temp/$bath)
225       if ($critical >  10. ) then
226          display  ****&&&& rerun job with smaller timestep (i.e., 0.003)
227          stop
228       end if
229
230    end loop cool
231
232    {* ================================================ Final minimization.*}
233
234    minimize powell nstep= 200 nprint=25 end
235
236     {* ====================== Analyze and write out the final structure(s).*}
```

```
   print threshold=0.5 noe
   evaluate ($rms_noe=$result)
   evaluate ($violations_noe=$violations)
   print threshold=5. cdih
   evaluate ($rms_cdih=$result)
   evaluate ($violations_cdih=$violations)
   print thres=0.05 bonds
   evaluate ($rms_bonds=$result)
   print thres=5. angles
   evaluate ($rms_angles=$result)
   print thres=5. impropers
   evaluate ($rms_impropers=$result)
   remarks ============================================================
   remarks            overall,bonds,angles,improper,vdw,noe,cdih
   remarks energies: $ener, $bond, $angl, $impr, $vdw, $noe, $cdih
   remarks ============================================================
   remarks            bonds,angles,impropers,noe,cdih
   remarks rms-d: $rms_bonds,$rms_angles,$rms_impropers,$rms_noe,$rms_cdih
   remarks ============================================================
   remarks               noe,  cdih
   remarks violations.: $violations_noe, $violations_cdih
   remarks ============================================================
   remarks handedness: $hand, enantiomer discrimination ( $e_plus : $e_minus )

{====>}                        {*Name(s) of the family of final structures.*}
    evaluate ($filename="dgsa_"+encode($count)+".pdb")

   write coordinates output =$filename end

end loop main

stop
```

20.3.4 Full-Structure Distance Geometry

This example input file shows how to produce a family of embedded coordinates using distance geometry that starts with regularized substructures. If metrization is desired, the appropriate lines should be uncommented. The names of the family of embedded coordinates will be "dg_full_embed_1.pdb" through "dg_full_embed_10.pdb". Although the "dg_full_embed.inp" protocol contains some regularization of the coordinates, further regularization is required by the "dgsa.inp" simulated annealing protocol.

Disulfide bonds and other covalent links should be treated in the same fashion as in the prior template generation. The removed covalent links are reintroduced as distance restraints.

```
remarks file nmr/dg_full_embed.inp --
remarks           Metrization and full embedding to produce a
remarks           family of DG-structures.
remarks Author: Axel Brunger

{====>}
structure @g_protein.psf end                          {*Read structure file.*}


parameter
{====>}
   @TOPPAR:parallhdg.pro                                 {*Read parameters.*}
   nbonds
      repel = 0.8                    {*This scales the van der Waals radii.*}
   end
```

```
end

topology
   presidue NDIS
      delete bond 1SG 2SG
      delete angle 1CB 1SG 2SG
      delete angle 1SG 2SG 2CB
   end
end

{===>}
               {*If protein contains S-S bridges, appropriately modify and *}
               {*then uncomment the following lines. The S-S covalent bonds*}
               {*will be deleted and reintroduced as fake NOE distances.   *}
!patch ndis reference=1=( residue 5  ) reference=2=( residue 55 ) end
!patch ndis reference=1=( residue 14 ) reference=2=( residue 38 ) end
!patch ndis reference=1=( residue 30 ) reference=2=( residue 51 ) end


noe
{====>}
   nres=3000                          {*Approximate number greater than the*}
                                      {*actual number of NOEs.             *}
   class = all

{====>}
   @g_protein_noe.tbl                       {*Read NOE distance ranges.   *}
                                            {*Note that no other settings *}
                                            {*are important for embedding.*}
end

{====>}
@g_protein_dihe.tbl                    {*Read dihedral angle restraints.*}


{====>}
               {*If protein contains S-S bridges, appropriately modify and *}
               {*then uncomment the following lines.                       *}
!noe
!   assign (resid  5 and name sg) (resid 55 and name sg)  2.02 0.1 0.1
!   assign (resid  14 and name sg) (resid 38 and name sg)  2.02 0.1 0.1
!   assign (resid 30 and name sg) (resid 51 and name sg)  2.02 0.1 0.1
!end


            {*Read template for pseudoatom correction and for target values*}
            {*for conformational constraints (bonds, angles, etc.).        *}
{===>} coor disp=refe @generate_template.pdb


evaluate ($count = 0)

                  {*The following loop produces a family of 10 structures.*}
{====>}
while ($count < 10 ) loop main

   evaluate ($count=$count+1)
      evaluate ($embedded = false)

   coor initialize end

{====>}                {*Read a regularized substructure if required.    *}
                       {*If no substructures are present, comment out the*}
                       {*next two lines.                                  *}
   evaluate ($filename="dg_sub_embed_"+encode($count)+".pdb")
   coor @@$filename

                                    {*Energy flags: both NOEs and dihedral*}
                                    {*angle restraints are included,      *}
   flags
```

```
86       exclude * include bond angle dihedral improper vdw noe cdih
87    end
88
89
90    mmdg
91       reference=coordinates
92
93       group ( known ) 0.5                          {*Substructure information.*}
94
95       selection=( all )                               {*Specify parameters    *}
96       scale=10. exponent=2                            {*for DG-regularization.*}
97
98       storebounds                              {*Store bounds matrix in memory.*}
99    end
100                                          {*Loop until embedding is successful;*}
101                                          {*normally the success rate is high, *}
102                                          {*and it will work during the first  *}
103                                          {*pass.                              *}
104   while ($embedded = false) loop embed
105      mmdg
106         recallbounds
107{====>}                          {*Comment out the next two statements if no *}
108                                  {*metrization is requested.                 *}
109         random
110         metrization=( all )=4
111      end
112   end loop embed
113
114   vector do (x = x * $dgscale) (known)              {*Scale the structure;*}
115   vector do (y = y * $dgscale) (known)              {*the symbol $DGSCALE *}
116   vector do (z = z * $dgscale) (known)              {*is defined by mmdg. *}
117
118                                        {*Include DG-term for regularization.*}
119   flags exclude * include dg end
120
121   minimize powell                                             {*Regularization.*}
122      nstep=100 drop=1. nprint=25
123   end
124
125                                           {*Write the embedded coordinates.*}
126
127   remarks  produced by nmr/dg_full_embed.inp
128
129{====>}                      {*Name(s) of the family of embedded structures.*}
130   evaluate ($filename="dg_full_embed_"+encode($count)+".pdb")
131
132   write coordinates output =$filename end
133
134end loop main
135
136stop
```

20.4 *Ab Initio* SA Starting from the Template

The example input file below shows how to produce a family of coordinates using *ab initio* simulated annealing starting from the template coordinate set (Nilges et al. 1991). The example file is general and should work for all types of macromolcules, although it has been tested only for proteins up to 2400 atoms. Important parameters that the user might want to change are the simulated annealing starting temperature, the duration of the high-temperature stage, and the duration of the cooling stage. The protocol works well for proteins with up to about 1000 atoms. For 2000 atoms, the durations

should be multiplied by a factor of about four. The protocol makes use of the soft-square potential (Section 18.3.3, Nilges et al. 1988c). The filenames of the family of embedded coordinates are "sa_1.pdb" through "sa_10.pdb". The coordinate files contain useful information about the energies, restraint-satisfaction, and violations.

```
remarks file  nmr/sa.inp
remarks  Simulated annealing protocol for NMR structure determination.
remarks  The starting structure for this protocol can be any structure with
remarks  a reasonable geometry, such as randomly assigned torsion angles or
remarks  extended strands.
remarks  Author: Michael Nilges

{====>}
evaluate ($init_t = 1000 )       {*Initial simulated annealing temperature.*}
{====>}
evaluate ($high_steps= 6000 )         {*Total number of steps at high temp.*}
{====>}
evaluate ($cool_steps = 3000 )      {*Total number of steps during cooling.*}

parameter                                        {*Read the parameter file.*}
{====>}
   @TOPPAR:parallhdg.pro
end

{====>}
structure @g_protein.psf end                     {*Read the structure file.*}

{====>}
coordinates @generate_template.pdb                  {*Read the coordinates.*}


noe
{====>}
   nres=3000             {*Estimate greater than the actual number of NOEs.*}
   class all
{====>}
   @g_protein_noe.tbl                           {*Read NOE distance ranges.*}
end

{====>}
@g_protein_dihe.tbl                       {*Read dihedral angle restraints.*}


{====>}
               {*If protein contains S-S bridges, appropriately modify and *}
               {*then uncomment the following lines. The S-S covalent bonds*}
               {*will be represented as fake NOE distances.                *}
noe
!   assign (resid  5 and name sg) (resid 55 and name sg)  2.02 0.1 0.1
!   assign (resid  14 and name sg) (resid 38 and name sg)  2.02 0.1 0.1
!   assign (resid 30 and name sg) (resid 51 and name sg)  2.02 0.1 0.1
end


flags exclude * include bonds angle impr vdw noe cdih end

                        {*Friction coefficient for MD heatbath, in 1/ps.   *}
vector do (fbeta=10) (all)
                        {*Uniform heavy masses to speed molecular dynamics.*}
vector do (mass=100) (all)

noe                               {*Parameters for NOE effective energy term.*}
  ceiling=1000
  averaging  * cent
  potential  * soft
  scale      * 50.
  sqoffset   * 0.0
```

```
  sqconstant * 1.0
  sqexponent * 2
  soexponent * 1
  asymptote  * 0.1                          {*Initial value--modified later.*}
  rswitch    * 0.5
end

parameter                          {*Parameters for the repulsive energy term.*}
    nbonds
      repel=1.                      {*Initial value for repel--modified later.*}
      rexp=2 irexp=2 rcon=1.
      nbxmod=3
      wmin=0.01
      cutnb=4.5 ctonnb=2.99 ctofnb=3.
      tolerance=0.5
   end
end

restraints dihedral
      scale=5.
end

{====>}
evaluate ($end_count=10)          {*Loop through a family of 10 structures.*}

coor copy end

evaluate ($count = 0)
while ($count < $end_count ) loop main

   evaluate ($count=$count+1)

   coor swap end
   coor copy end

     {* ============================================= Initial minimization.*}
   restraints dihedral   scale=5.   end
   noe asymptote * 0.1  end
   parameter  nbonds repel=1.   end end
   constraints interaction
           (not name SG) (all) weights * 1  vdw 0.002 end end
   minimize powell nstep=50 drop=10.  nprint=25 end


     {* ======================================== High-temperature dynamics.*}
   constraints interaction (not name SG) (all)
               weights * 1  angl 0.4  impr 0.1 vdw 0.002 end end

   evaluate ($nstep1=int($high_steps * 2. / 3. ) )
   evaluate ($nstep2=int($high_steps * 1. / 3. ) )

   dynamics  verlet
      nstep=$nstep1   timestep=0.005   iasvel=maxwell   firstt=$init_t
      tcoupling=true  tbath=$init_t  nprint=50  iprfrq=0
   end


     {* ============== Tilt the asymptote and increase weights on geometry.*}
   noe asymptote * 1.0  end

   constraints interaction
             (not name SG) (all) weights * 1  vdw 0.002  end end
   dynamics  verlet
      nstep=$nstep2   timestep=0.005    iasvel=current   tcoupling=true
      tbath=$init_t  nprint=50  iprfrq=0
   end

    {* ================================================  Cool the system.*}

   restraints dihedral   scale=200.   end
```

```
133
134    evaluate ($final_t = 100)      { K }
135    evaluate ($tempstep = 50)      { K }
136
137    evaluate ($ncycle = ($init_t-$final_t)/$tempstep)
138    evaluate ($nstep = int($cool_steps/$ncycle))
139
140    evaluate ($ini_rad  = 0.9)        evaluate ($fin_rad  = 0.75)
141    evaluate ($ini_con=  0.003)       evaluate ($fin_con=  4.0)
142
143    evaluate ($bath  = $init_t)
144    evaluate ($k_vdw = $ini_con)
145    evaluate ($k_vdwfact = ($fin_con/$ini_con)^(1/$ncycle))
146    evaluate ($radius=    $ini_rad)
147    evaluate ($radfact = ($fin_rad/$ini_rad)^(1/$ncycle))
148
149    evaluate ($i_cool = 0)
150    while ($i_cool < $ncycle) loop cool
151       evaluate ($i_cool=$i_cool+1)
152
153       evaluate ($bath  = $bath  - $tempstep)
154       evaluate ($k_vdw=min($fin_con,$k_vdw*$k_vdwfact))
155       evaluate ($radius=max($fin_rad,$radius*$radfact))
156
157       parameter  nbonds repel=$radius   end end
158       constraints interaction (not name SG) (all)
159                     weights * 1. vdw $k_vdw end end
160
161       dynamics  verlet
162          nstep=$nstep time=0.005 iasvel=current firstt=$bath
163          tcoup=true tbath=$bath nprint=$nstep iprfrq=0
164       end
165
166
167 {====>}                                                {*Abort condition.*}
168       evaluate ($critical=$temp/$bath)
169       if ($critical >  10. ) then
170          display  ****&&&& rerun job with smaller timestep (i.e., 0.003)
171          stop
172       end if
173
174    end loop cool
175
176    {* ================================================ Final minimization.*}
177
178    constraints interaction (all) (all) weights * 1. vdw 4. end end
179    minimize powell nstep=200 drop=10.0 nprint=25 end
180
181    {* ================================== Write out the final structure(s).*}
182    print threshold=0.5 noe
183    evaluate ($rms_noe=$result)
184    evaluate ($violations_noe=$violations)
185    print threshold=5. cdih
186    evaluate ($rms_cdih=$result)
187    evaluate ($viscations_cdih=$violations)
188    print thres=0.05 bonds
189    evaluate ($rms_bonds=$result)
190    print thres=5. angles
191    evaluate ($rms_angles=$result)
192    print thres=5. impropers
193    evaluate ($rms_impropers=$result)
194    remarks ============================================================
195    remarks            overall,bonds,angles,improper,vdw,noe,cdih
196    remarks energies: $ener, $bond, $angl, $impr, $vdw, $noe, $cdih
197    remarks ============================================================
198    remarks            bonds,angles,impropers,noe,cdih
199    remarks rms-d: $rms_bonds,$rms_angles,$rms_impropers,$rms_noe,$rms_cdih
200    remarks ============================================================
201    remarks              noe,  cdih
202    remarks violations.: $violations_noe, $violations_cdih
```

```
203    remarks ============================================================
204
205 {====>}                        {*Name(s) of the family of final structures.*}
206    evaluate ($filename="sa_"+encode($count)+".pdb")
207
208    write coordinates output =$filename end
209
210 end loop main
211
212 stop
```

In order to improve the sampling of this protocol it might be advisable to use different template structures or to randomize the torsion angles of the template structure. The file "../nmr/randomchain.inp" shows how to randomize the ϕ and ψ angles of a template structure for a protein.

20.5 Random Simulated Annealing

The example input file below shows how to produce a family of coordinates using simulated annealing starting from random coordinates (Nilges et al. 1988a). The example file is general and should work for all types of macromolecules, although it has been tested only for proteins of up to 2400 atoms. The user might want to change the simulated annealing starting temperature if convergence is unsatisfactory. The duration of the high-temperature stage is internally determined. The filenames of the family of coordinates obtained by this protocol are "random_1.pdb" through "random_10.pdb". These coordinates usually require extensive regularization by using the "dgsa.inp" protocol (Section 20.3.3). Note that the "dgsa.inp" protocol requires the template coordinates even though the "random.inp" does not.

```
1  remarks nmr/random.inp
2  remarks  The ultimate simulated annealing protocol for NMR structure
3  remarks  determination!
4  remarks  The starting structure for this protocol can be completely
5  remarks  arbitrary, such as random numbers.  Note: the resulting
6  remarks  structures need to be further processed by the dgsa.inp protocol.
7  remarks  Reference: Nilges, Clore, and Gronenborn 1988a.
8  remarks Author: Michael Nilges
9
10 {====>}
11 evaluate ($init_t = 1000 )      {* Initial simulated annealing temperature.*}
12
13
14 parameter                                        {*Read the parameter file.*}
15 {====>}
16    @TOPPAR:parallhdg.pro
17 end
18
19 {====>}
20 structure @g_protein.psf end                     {*Read the structure file.*}
21
22 noe
23 {====>}
24    nres=3000             {*Estimate greater than the actual number of NOEs.*}
25    class all
26 {====>}
27    @g_protein_noe.tbl                          {*Read NOE distance ranges.*}
28 end
```

```
29
30 {====>}
31 @g_protein_dihe.tbl                        {*Read dihedral angle restraints.*}
32
33
34 noe                               {*Parameters for NOE effective energy term.*}
35   ceiling=1000
36   averaging  * cent
37   potential  * soft
38   scale      * 1
39   sqoffset   * 0.0
40   sqconstant * 1.0
41   sqexponent * 2
42   soexponent * 1
43   asymptote  * 2.0                      {* Initial value - modified later. *}
44   rswitch    * 1.0
45 end
46
47 set message=off end
48
49 {====>} evaluate ($end_count=10)                  {* Number of structures. *}
50
51 evaluate ($count = 0)
52 while ($count < $end_count ) loop main
53     evaluate ($count=$count+1)
54
55                                          {* Generate a starting structure. *}
56     vector do (x = (random()-0.5)*20) (all)
57     vector do (y = (random()-0.5)*20) (all)
58     vector do (z = (random()-0.5)*20) (all)
59
60     vector do (fbeta=10) (all)      {*Friction coefficient for MD heatbath.*}
61     vector do (mass=100) (all)              {*Uniform heavy masses to speed*}
62                                             {*molecular dynamics.          *}
63
64     parameter nbonds
65          atom cutnb 100 tolerance 45 repel=1.2
66          rexp=2 irexp=2 rcon=1.0 nbxmod 4
67     end end
68
69     flags exclude * include bonds angle impr vdw noe cdih  harm end
70
71     evaluate ($knoe = 0.5)
72
73     evaluate ($kbon = 0.00005  )                {* Bonds.                 *}
74     evaluate ($kang = 0.00005  )                {* Angles.                *}
75     evaluate ($kimp = 0.0  )                    {* Impropers.             *}
76     evaluate ($kvdw = 0.1)                      {* Vdw.                   *}
77     constraints
78         interaction (not name ca) (all)
79         weights bond $kbon angl $kang impr $kimp vdw 0 elec 0 end
80         interaction (name ca) (name ca)
81         weights bond $kbon angl $kang impr $kimp vdw $kvdw end
82     end
83
84
85     {* ======================================== High temperature dynamics. *}
86
87     vector do (vx = maxwell($init_t)) (all)
88     vector do (vy = maxwell($init_t)) (all)
89     vector do (vz = maxwell($init_t)) (all)
90
91     evaluate ($timestep = 0.04)
92     evaluate ($nstep = 100)
93
94     while ($kbon < 0.01) loop stage1
95
96             evaluate ($kbon = min(0.25, $kbon * 1.25))
97             evaluate ($kang = $kbon)
98             evaluate ($kimp = 0)
```

```
99
100             noe scale * $knoe end
101             restraints dihed scale 0. end
102             constraints
103                 interaction (not name ca) (all)
104                 weights bond $kbon angl $kang impr $kimp vdw 0 elec 0 end
105                 interaction (name ca) (name ca)
106                 weights bond $kbon angl $kang impr $kimp vdw $kvdw end
107             end
108
109             dynamics  verlet
110                   nstep=$nstep   timestep=$timestep   iasvel=current
111                   tcoupling=true tbath=$init_t  nprint=50  iprfrq=0
112             end
113
114     end loop stage1
115
116     restraints dihed scale 0. end
117     noe scale * 5 end
118     parameter               {* Parameters for the repulsive energy term.  *}
119       nbonds
120         repel=0.9           {* Initial value for repel - modified later.  *}
121         nbxmod=-3           {* Initial value for nbxmod - modified later. *}
122         wmin=0.01
123         cutnb=4.5 ctonnb=2.99 ctofnb=3.
124         tolerance=0.5
125       end
126     end
127     constraints
128                 interaction (all) (all)
129                 weights bond 0.02 angl 0.02 impr 0 vdw 0.002 elec 0 end
130     end
131     dynamics  verlet
132                 nstep=500   timestep=0.003   iasvel=maxwell
133                 firstt=1500
134                 tcoupling=true
135                 tbath=1500  nprint=50  iprfrq=0
136     end
137     constraints
138                 interaction (all) (all)
139                 weights bond 0.05 angl 0.05 impr 0 vdw 0.005 elec 0 end
140     end
141     dynamics  verlet
142                   nstep=500   timestep=0.003   iasvel=current  tcoupling=true
143                   tbath=1500  nprint=50  iprfrq=0
144     end
145     constraints
146                 interaction (all) (all)
147                 weights bond 0.1 angl 0.1 impr 0 vdw 0.01 elec 0 end
148     end
149     dynamics  verlet
150                   nstep=500   timestep=0.003   iasvel=current  tcoupling=true
151                   tbath=1500  nprint=50  iprfrq=0
152     end
153
154    {* ================================== Write out the final structure(s).*}
155    remarks ============================================================
156    remarks            overall,bonds,angles,improper,vdw,noe,cdih
157    remarks energies: $ener, $bond, $angl, $impr, $vdw, $noe, $cdih
158
159
160 {===>}                        {*Name(s) of the family of final structures.*}
161    evaluate ($filename="random_"+encode($count)+".pdb")
162
163    write coordinates output =$filename end
164
165 end loop main
166
167 stop
```

20.6 Simulated Annealing Refinement

The example input files below shows how to refine NMR-derived structures produced by any method inside or outside X-PLOR. The "refine.inp" protocol is of the slow-cooling type reminiscent of the protocol used in crystallographic refinement (see Section 13.1.3). The additional feature included in this protocol is a softening of the van der Waals repulsions. This enables atoms to move through each other. The "refine_gentle.inp" protocol accomplishes the refinement of the coordinates by less dramatic modifications of the energy function. It is intended for under-determined systems where the full empirical energy function is required to determine the final structure of the macromolecule. This applies in particular to structures of short oligonucleotide fragments. Note that the "refine_gentle.inp" protocol uses the standard Lennard-Jones function, electrostatic interactions, and dihedral terms in addition to the conformational energy terms used in the "refine.inp" protocol.

The "refine.inp" protocol is shown below. Important parameters that the user might want to change are the simulated annealing starting temperature and the duration of the cooling stage. The filenames of the family of embedded coordinates are "refine_1.pdb" through "refine_10.pdb". The coordinate files contain useful information about the energies, restraint-satisfaction, and violations. Further refinement can be accomplished by repeated application of this protocol (note that the names of the input and output coordinate sets have to be changed) or by an increase of the cooling period. Repeated application is generally more efficient, because it allows the user to inspect periodically the restraint-statisfaction in the family of refined coordinates and to remove ill-behaved coordinates. This removal of ill-behaved coordinates can be automated by using the acceptance protocol described in the next section.

```
1  remarks file nmr/refine.inp  -- Simulated annealing refinement
2  remarks                      for NMR structure determination
3  remarks  Authors: Michael Nilges, John Kuszewski, and Axel T. Brunger
4
5  {====>}
6  evaluate ($init_t = 1000)           {*Initial annealing temperature, in K.*}
7  {====>}
8  evaluate ($cool_steps = 2000 )      {*Total number of steps during cooling.*}
9
10 parameter                                        {*Read the parameter file.*}
11 {====>}
12    @TOPPAR:parallhdg.pro
13 end
14
15 {====>} structure @g_protein.psf end                  {*The structure file.*}
16
17 noe
18 {====>}
19    nres=3000             {*Estimate greater than the actual number of NOEs.*}
20    class all
21 {====>}
22    @g_protein_noe.tbl                          {*Read NOE distance ranges.*}
23 end
24
```

```
{====>}
@g_protein_dihe.tbl                      {*Read dihedral angle restraints.*}

                           {*Friction coefficient for MD heatbath, in 1/ps.*}
vector do (fbeta=10) (all)
vector do (mass=100) (all)      {*Heavy masses to speed molecular dynamics.*}

noe                             {*Parameters for NOE effective energy term.*}
   ceiling=1000
   averaging  * cent
   potential  * square
   sqconstant * 1.
   sqexponent * 2
   scale      * 50.           {*Constant NOE scale throughout the protocol.*}
end

parameter                       {*Parameters for the repulsive energy term.*}
   nbonds
      repel=0.5                  {*Initial value for repel--modified later.*}
      rexp=2 irexp=2 rcon=1.
      nbxmod=3
      wmin=0.01
      cutnb=4.5 ctonnb=2.99 ctofnb=3.
      tolerance=0.5
   end
end

restraints dihedral
       scale=200.
end

{====>}
evaluate ($end_count=10)          {*Loop through a family of 10 structures.*}

evaluate ($count = 0)
while ($count < $end_count ) loop main

   evaluate ($count=$count+1)

{====>}                             {*Filename(s) for embedded coordinates.*}
   evaluate ($filename="dgsa_"+encode($count)+".pdb")

   coor @@$filename

   flags exclude * include bond angl impr vdw noe cdih end

   vector do (vx=maxwell($init_t)) ( all )
   vector do (vy=maxwell($init_t)) ( all )
   vector do (vz=maxwell($init_t)) ( all )

   evaluate ($final_t = 100)     { K }
   evaluate ($tempstep = 50)     { K }

   evaluate ($ncycle = ($init_t-$final_t)/$tempstep)
   evaluate ($nstep = int($cool_steps/$ncycle))

   evaluate ($ini_rad  = 0.9)        evaluate ($fin_rad  = 0.75)
   evaluate ($ini_con=  0.003)       evaluate ($fin_con=  4.0)

   evaluate ($bath  = $init_t)
   evaluate ($k_vdw = $ini_con)
   evaluate ($k_vdwfact = ($fin_con/$ini_con)^(1/$ncycle))
   evaluate ($radius=    $ini_rad)
   evaluate ($radfact = ($fin_rad/$ini_rad)^(1/$ncycle))

   evaluate ($i_cool = 0)
   while ($i_cool < $ncycle) loop cool
      evaluate ($i_cool=$i_cool+1)

      evaluate ($bath  = $bath  - $tempstep)
```

```
95       evaluate ($k_vdw=min($fin_con,$k_vdw*$k_vdwfact))
96       evaluate ($radius=max($fin_rad,$radius*$radfact))
97
98       parameter  nbonds repel=$radius   end end
99       constraints interaction (all) (all) weights * 1. vdw $k_vdw end end
100
101      dynamics  verlet
102         nstep=$nstep time=0.005 iasvel=current firstt=$bath
103         tcoup=true tbath=$bath nprint=$nstep iprfrq=0
104      end
105
106
107 {====>}                                                  {*Abort condition.*}
108      evaluate ($critical=$temp/$bath)
109      if ($critical >  10. ) then
110         display  ****&&&& rerun job with smaller timestep (i.e., 0.003)
111         stop
112      end if
113   end loop cool
114
115   {* ================================================ Final minimization.*}
116   minimize powell nstep= 200 nprint=25 end
117
118   {* ================================== Write out the final structure(s).*}
119   print threshold=0.5 noe
120   evaluate ($rms_noe=$result)
121   evaluate ($violations_noe=$violations)
122   print threshold=5. cdih
123   evaluate ($rms_cdih=$result)
124   evaluate ($violations_cdih=$violations)
125   print thres=0.05 bonds
126   evaluate ($rms_bonds=$result)
127   print thres=5. angles
128   evaluate ($rms_angles=$result)
129   print thres=5. impropers
130   evaluate ($rms_impropers=$result)
131   remarks ============================================================
132   remarks            overall,bonds,angles,improper,vdw,noe,cdih
133   remarks energies: $ener, $bond, $angl, $impr, $vdw, $noe, $cdih
134   remarks ============================================================
135   remarks            bonds,angles,impropers,noe,cdih
136   remarks rms-d: $rms_bonds,$rms_angles,$rms_impropers,$rms_noe,$rms_cdih
137   remarks ============================================================
138   remarks               noe,  cdih
139   remarks violations.: $violations_noe, $violations_cdih
140   remarks ============================================================
141
142 {====>}                        {*Name(s) of the family of final structures.*}
143   evaluate ($filename="refine_"+encode($count)+".pdb")
144
145   write coordinates output =$filename end
146
147 end loop main
148
149 stop
```

The "refine_gentle.inp" protocol is show below. In this case two initial DNA structures ("dna_init_1.pdb" and "dna_init_2.pdb") containing ideal A and B–type conformations are refined. Apart from the experimental NOEs, distance restraints for the base-pair hydrogen bonds are included. Note that one can also include planarity restraints (Section 7.3) to maintain the planarity of the base pairs. Average coordinates for the last 10 psec of the molecular dynamics calculation are computed and then refined by conjugate gradient minimization.

```
remarks file nmr/refine_gentle.inp  -- Gentle simulated annealing refinement
remarks                                for NMR structure determination

{====>}
evaluate ($init_t = 300)                              {*SA temperature, in K.*}
{====>}
evaluate ($cool_steps = 20000 )                     {*Total number of SA steps.*}


parameter                                            {*Read the parameter file.*}
{====>}
   @TOPPAR:parallhdg.dna
end

{====>} structure @generatedna.psf end              {*Read the structure file.*}

noe
{====>}
   nres=3000             {*Estimate greater than the actual number of NOEs.*}
   class all
{====>}
   @inter2.tbl                                      {*Read NOE distance ranges.*}
   @intra2.tbl
   @hbonds.tbl
end

{====>}
@dihed.tbl                                  {*Read dihedral angle restraints.*}


noe                                {*Parameters for NOE effective energy term.*}
   ceiling=1000
   averaging  * cent
   potential  * square
   sqconstant * 1.
   sqexponent * 2
   scale      * 50.            {*Constant NOE scale throughout the protocol.*}
end

parameter                          {*Parameters for the repulsive energy term.*}
   nbonds
      tolerance=0.5
      cutnb=11.5 ctonnb=9.5 ctofnb=10.5 tolerance=0.5 rdie vswitch switch
   end
end

restraints dihedral
        scale=200.
end

{====>}
evaluate ($end_count=2)              {*Loop through a family of 2 structures.*}

evaluate ($count = 0)
while ($count < $end_count ) loop main

   evaluate ($count=$count+1)

{====>}                               {*Filename(s) for embedded coordinates.*}
   evaluate ($filename="dna_init_"+encode($count)+".pdb")

   coor @@$filename

   flags exclude * include bond angl impr vdw noe cdih elec dihe end


   {* ============================================== Intial minimization.*}
   minimize powell nstep= 200 nprint=25 end

   {* ============================================= Constant temperature SA.*}
```

```
71    vector do ( fbeta=100. ) ( all )              {* Coupling to heat bath. *}
72    dynamics  verlet
73       nstep=$cool_steps timestep=0.001
74       iasvel=maxwell  firsttemperature=$init_t
75       tbath=300
76       nprint=250 iprfrq=2500
77       trajectory=refine_gentle.tra   nsavc=200 {* Save trajectory to file. *}
78    end
79    close refine_gentle.tra end
80
81    {* ======================== Compute dynamics average over last 10 psec.*}
82    dynamics analysis average
83        input=refine_gentle.tra begin=10000 skip=200 stop=$cool_steps
84    end
85    close refine_gentle.tra end
86
87    {* ============================================== Final minimization.*}
88    minimize powell nstep= 200 nprint=25 end
89
90    {* ================================== Write out the final structure(s).*}
91    print threshold=0.5 noe
92    evaluate ($rms_noe=$result)
93    evaluate ($violations_noe=$violations)
94    print threshold=5. cdih
95    evaluate ($rms_cdih=$result)
96    evaluate ($violations_cdih=$violations)
97    print thres=0.05 bonds
98    evaluate ($rms_bonds=$result)
99    print thres=5. angles
100   evaluate ($rms_angles=$result)
101   print thres=5. impropers
102   evaluate ($rms_impropers=$result)
103   remarks ============================================================
104   remarks            overall,bonds,angles,improper,vdw,noe,cdih
105   remarks energies: $ener, $bond, $angl, $impr, $vdw, $noe, $cdih
106   remarks ============================================================
107   remarks            bonds,angles,impropers,noe,cdih
108   remarks rms-d: $rms_bonds,$rms_angles,$rms_impropers,$rms_noe,$rms_cdih
109   remarks ============================================================
110   remarks               noe,  cdih
111   remarks violations.: $violations_noe, $violations_cdih
112   remarks ============================================================
113
114 {====>}                       {*Name(s) of the family of final structures.*}
115   evaluate ($filename="refine_gentle_"+encode($count)+".pdb")
116
117   write coordinates output =$filename end
118
119 end loop main
120
121 stop
```

20.7 Acceptance of Refined NMR Structures

The example input file below shows how to analyze a family of refined NMR structures produced by any method inside or outside X-PLOR. The protocol also produces a subfamily of accepted coordinates with criteria that the user can determine. In the example file below, it is required that no NOE distances be outside the distance bounds plus 0.5 Å, no dihedral angle restraint violations be greater than 5°, and the geometry be very tight. This is typical for very good NMR structures. For less ideal systems, these criteria may have to be relaxed. The filenames of the family of accepted coordinates are

"accept_1.pdb" through "accept_10.pdb". (For analysis of other geometric and energetic properties, such as Ramachandran plots, see Chapter 5.)

```
remarks file  nmr/accept.inp
remarks  Analysis of a family of NMR structures--
remarks  generation of a subfamily of "acceptable" structures

parameter                                          {*Read the parameter file.*}
{====>}
   @TOPPAR:parallhdg.pro
end

{====>}
    structure @g_protein.psf end                   {*Read the structure file.*}

noe
{====>}
   nres=3000             {*Estimate greater than the actual number of NOEs.*}
   class all
{====>}
   @g_protein_noe.tbl                            {*Read NOE distance ranges.*}
end

{====>}
@g_protein_dihe.tbl                        {*Read dihedral angle restraints.*}

noe                               {*Parameters for NOE effective energy term.*}
   ceiling=1000
   averaging  * cent
   potential  * square
   sqconstant * 1.
   sqexponent * 2
   scale      * 50.
end

parameter                         {*Parameters for the repulsive energy term.*}
   nbonds
      repel=0.75
      rexp=2 irexp=2 rcon=4.
      nbxmod=3
      wmin=0.01
      cutnb=4.5 ctonnb=2.99 ctofnb=3.
      tolerance=0.5
   end
end

restraints dihedral
      scale=200.
end

flags exclude * include bonds angle impr vdw noe cdih end

set precision=4 end

{====>}
evaluate ($end_count=10)          {*Loop through a family of 10 structures.*}

evaluate ($accept_count = 0)
evaluate ($count = 0)
while ($count < $end_count ) loop main
   evaluate ($count=$count+1)
{====>}                             {*Filename(s) for embedded coordinates.*}
   evaluate ($filename="refine_"+encode($count)+".pdb")

   coor @@$filename

   evaluate ($accept=0)
                                {*Print all NOE violations larger than 0.3 A *}
                                {*and compute RMS difference between observed*}
```

```
                              {*and model distances.                        *}
   print threshold=0.5 noe
   evaluate ($rms_noe=$result)
   evaluate ($violations_noe=$violations)
   if ($violations_noe > 0) then  evaluate ( $accept=$accept + 1)   end if

                                      {*Print all dihedral angle restraint*}
                                      {*violations.                         *}
   print threshold=5. cdih
   evaluate ($rms_cdih=$result)
   evaluate ($violations_cdih=$violations)
   if ($violations_cdih > 0) then  evaluate ( $accept=$accept + 1)   end if

   print thres=0.05 bonds         {*Print deviations from ideal geometry.*}
   evaluate ($rms_bonds=$result)
   if ($result > 0.01) then  evaluate ( $accept=$accept + 1)   end if

   print thres=5. angles
   evaluate ($rms_angles=$result)
   if ($result > 1) then  evaluate ( $accept=$accept + 1)   end if

   print thres=5. impropers
   evaluate ($rms_impropers=$result)

   distance from=( not hydrogen ) to=( not hydrogen ) cutoff=1.5 end

               {*Acceptance criteria: no NOE violations greater than 0.5 A,*}
               {*no dihedral angle restraint violations > 5 deg,           *}
               {*rms difference for bond deviations from ideality < 0.01 A,*}
               {*rms difference for angle deviations from ideality < 2 deg.*}
   energy end

   if ($accept = 0 ) then
      evaluate ($accept_count=$accept_count+1)

{====>}
      evaluate ($filename="accept_"+encode($accept_count)+".pdb")
      remarks ============================================================
      remarks            overall,bonds,angles,vdw,noe,cdih
      remarks energies: $ener, $bond, $angl, $vdw, $noe, $cdih, $impr
      remarks ============================================================
      remarks            bonds, angles, impropers, noe, cdih
      remarks rms-d: $rms_bonds,$rms_angles,$rms_impropers,$rms_noe,$rms_cdih
      remarks ============================================================
      remarks               noe,  cdih
      remarks violations.: $violations_noe, $violations_cdih

      write coordinates output=$filename end
   end if

end loop main

stop
```

20.8 Average Structure and Rmsds

The example input file below shows how to compute average coordinates and rms differences (rmsds) from the mean for a family of coordinates. For the computation of the rmsds between independent coordinates (e.g., between an NMR-derived structure and a crystal structure) and the analysis of distance differences, the reader is referred to Chapter 6. The example file produces a new coordinate file ("average.pdb"), which contains the average

coordinates in the x,y,z positions and the rmsds in the b (B-factor) position of the PDB coordinate file. The remarks records of the coordinate file also contain information about the overall rmsd for both backbone atoms and nonhydrogen atoms.

```
remarks file  nmr/average.inp
remarks Computes the average structure, atomic rms differences from the
remarks mean for a family of structures, and average overall rms
remarks difference between the family and the mean structure.

{====>}
structure @g_protein.psf end                    {*Read the structure file.*}


{====>}       {*"Backbone" selection--this example is typical for proteins.*}
vector idend ( store9 ) ( name ca or name n or name c )


   {*============ The first stage consists of computing the mean structure.*}

{====>}                 {*Loop through the family of 8 accepted structures.*}
evaluate ($end_count=8)

eval ($nfile=0)
vector do (store1=0) (all)
vector do (store2=0) (all)
vector do (store3=0) (all)
vector do (store4=0) (all)

evaluate ($count = 0)
while ($count < $end_count ) loop main
   evaluate ($count=$count+1)

{====>}                    {*This is the name of the family of structures.*}
   evaluate ($filename="accept_"+encode($count)+".pdb")

   coor @@$filename

   if ($count=1) then
      coor copy end             {*Store first structure in comparison set.*}
   end if

   coor sele=( recall9 ) fit end

                      {* swap equivalent groups to minimize rms difference *}
   @../geomanal/rotares.inp

   vector do (store1=store1+x) (all)
   vector do (store2=store2+y) (all)
   vector do (store3=store3+z) (all)
   vector do (store4=store4+x*x+y*y+z*z) (all)
   eval ($nfile=$nfile+1)
end loop main

vector do (x = store1 / $nfile) (all)
vector do (y = store2 / $nfile) (all)
vector do (z = store3 / $nfile) (all)
vector do (bcomp=sqrt(max(0,store4/$nfile-(x**2+y**2+z**2)))) (all)


        {*The second stage consists of computing an overall rms difference.*}

evaluate ($ave_rmsd_all=0.)
evaluate ($ave_rmsd_back=0.)

coor copy end

evaluate ($count = 0)
```

```
while ($count < $end_count ) loop main
   evaluate ($count=$count+1)


{====>}                    {*This is the name of the family of structures.*}
   evaluate ($filename="accept_"+encode($count)+".pdb")

   coor @@$filename
   coor fit sele=( recall9 ) end
   coor rms selection=( recall9 )end
   evaluate ($ave_rmsd_back=$ave_rmsd_back + $result)
   coor rms selection=( not hydrogen )end
   evaluate ($ave_rmsd_all =$ave_rmsd_all + $result)
end loop main

evaluate ($ave_rmsd_back=$ave_rmsd_back / $nfile)
evaluate ($ave_rmsd_all =$ave_rmsd_all  / $nfile)
display ave. rms diff. to the mean struct. for non-h atoms= $ave_rmsd_all
display ave. rms diff. to the mean struct. for the backbone= $ave_rmsd_back

   {*====== Finally, the average structure and RMSDs are written to a file.*}
coor swap end
vector do (b=bcomp) ( all )

remarks unminimized average over $nfile files
remarks ave. rms diff. to the mean struct. for non-h atoms= $ave_rmsd_all
remarks ave. rms diff. to the mean struct. for the backbone= $ave_rmsd_back
remarks b array (last column) is the rms difference from the mean

{====>}            {*Write average coordinates and RMSDs to specified file.*}
write coordinates output=average.pdb end


stop
```

Please note the use of the "rotares.inp" file that fits groups of equivalent atoms that are indistinguishable in NMR experiments (cf. Section 6.3). The input file below shows how to plot the rmsds provided by the "average.pdb" coordinate file:

```
remarks file nnr/average_plot.inp -- Rmsds per residue
remarks Specific for proteins: backbone and side-chain atoms.

{====>}                                              {*Read structure file.*}
structure @g_protein.psf end

{====>}
coordinates @average.pdb                               {*Read coordinates.*}

{====>}
vector ident (store9) ( tag )             {*This selects all residues to be*}
                                          {*analyzed.  Change it           *}
                                          {*to analyze portions,           *}
                                          {*e.g., ( tag and segid "A" );   *}
                                          {*"tag" assigns one unique atom * }
                                          {*per residue  (see <selection>).*}


{====>}
set display=average_plot.list end
                                        {*Write the plot information to the*}
                                        {*specified file.                  *}

set echo=off end                          {*Turn off echo to reduce output.*}
set message=off end                            {*Turn off warning messages.*}

evaluate ($number=0)
```

```
for $atom_id in id ( store9 ) loop out2
   vector show norm ( b ) ( byresidue ( id $atom_id )
                            and ( name ca or name n or name c ) )
   evaluate ( $back=$result )
   vector show norm ( b ) ( byresidue ( id $atom_id )
                            and not ( name ca or name n or name c )
                            and not hydrogen )
   evaluate ( $side=$result )
   vector show elem ( resid ) ( id $atom_id )

   evaluate ($number=$number+1)
   display   $number $back $side
end loop out2

set echo=on end
set message=on end

stop
```

The resulting text file "average_plot.list" can be plotted by Mathematica using the following script file. The result is shown in Fig. 20.2.

```
           (* file nmr/average_plot.math  -- Make a plot rmsd vs residue   *)

Off[General::spell1];
$DefaultFont={"Times-Roman",13};

                                      (* Change the name of the input file. *)
 (*===>*)    data=ReadList["average_plot.list",{Number,Number,Number}];

     column1 = Transpose[data][[1]];
     column2 = Transpose[data][[2]];
     column3 = Transpose[data][[3]];
     max2=Max[column2];
     max3=Max[column3];
     number=Dimensions[column1][[1]];
     ListPlot[column2,
      Axes->True,
      AxesLabel->{"Seq. Res. No.","rmsd / A"},
      Ticks->{Range[0, number , 5],Range[0.0, max2 ,0.2]},
      PlotLabel->"backbone", AxesOrigin->{0,0},PlotJoined->True,
      PlotRange->{{0, number+ 0.2 }, {0., max2 +0.2 }}];
     ListPlot[column3,
      Axes->True,
      AxesLabel->{"Seq. Res. No.","rmsd / A"},
      Ticks->{Range[0, number , 5],Range[0.0, max3 ,0.5]},
      PlotLabel->"side chain", AxesOrigin->{0,0},PlotJoined->True,
      PlotRange->{{0, number +0.2 }, {0., max3 +0.2 }}];


```

20.9 Pairwise Rmsds

The example input file below shows how to compute averaged pairwise rmsds for a family of coordinates. The example produces a text file ("rms_pairwise.txt") that contains the average pairwise rmsds for both backbone and nonhydrogen atoms.

```
remarks file  nmr/rms_pairwise.inp
remarks Computes the average pairwise rms difference for a family of
remarks structures

```

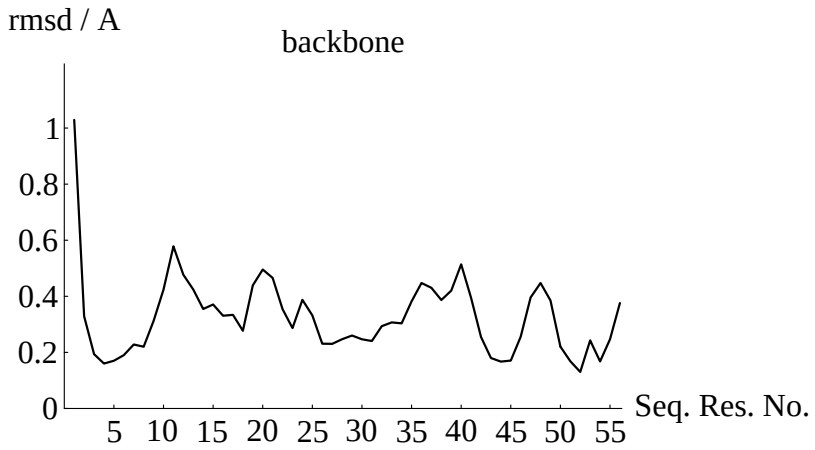

Figure 20.2: Rmsds for NMR structures.

```
{====>}
structure @g_protein.psf end                    {*Read the structure file.*}

{====>}       {*"Backbone" selection--this example is typical for proteins.*}
vector idend ( store9 ) ( name ca or name n or name c )


eval ($s1 = 0)
eval ($s2 = 0)
eval ($ss1 = 0)
eval ($ss2 = 0)
eval ($ncomp=0)
eval ($min1=9999)
eval ($min2=9999)
eval ($max1=0)
eval ($max2=0)

{====>}                 {*Loop through the family of 8 accepted structures.*}
evaluate ($end_count=8)

set message=off end
set echo=off end

evaluate ($count1 = 0)
while ($count1 < $end_count ) loop fil1
   evaluate ($count1=$count1+1)

                            {*This is the name of the family of structures.*}
{====>}
   evaluate ($file1="accept_"+encode($count1)+".pdb")

   coor disp=comp @@$file1

   evaluate ($count2 = 0)
   while ($count2 < $end_count ) loop fil2
   evaluate ($count2=$count2+1)

      if ($count1 < $count2) then
                            {*This is the name of the family of structures.*}
{====>}
         evaluate ($file2="accept_"+encode($count2)+".pdb")
         coor @@$file2
         coor sele=(recall 9) fit end
```

```
48          coor sele=(recall 9) rms end
49          eval ($b1 = $result)
50          coor sele=(not hydro) rms end
51          eval ($b2 = $result)
52
53          eval ($s1 = $s1 + $b1)
54          eval ($s2 = $s2 + $b2)
55          eval ($ss1 = $ss1 + $b1*$b1)
56          eval ($ss2 = $ss2 + $b2*$b2)
57          eval ($ncomp=$ncomp+1)
58          eval ($min1=min($b1,$min1))
59          eval ($min2=min($b2,$min2))
60          eval ($max1=max($b1,$max1))
61          eval ($max2=max($b2,$max2))
62       end if
63    end loop fil2
64 end loop fil1
65
66 set message=on end
67 set echo=on end
68                                {*Evaluate mean value and standard deviation.*}
69 eval ($s1=$s1/$ncomp)
70 eval ($s2=$s2/$ncomp)
71 eval ($ss1= sqrt(($ss1-$ncomp*$s1*$s1)/$ncomp))
72 eval ($ss2= sqrt(($ss2-$ncomp*$s2*$s2)/$ncomp))
73
74 {====>}
75 set display=rms_pairwise.txt end              {*Write result to text file.*}
76
77 display  pairwise rmsd analysis for a family of $end_count structures
78 display  backbone pairwise rmsd:
79 display      average=$s1, stand. dev.=$ss1, min= $min1, max= $max1
80 display  non-hydrogen atom pairwise rmsd:
81 display      average=$s2, stand. dev.=$ss2, min= $min2, max= $max2
82
83 stop
```

20.10 Time-Average Refinement

The following input file implements the time-averaged refinement procedure described in Section 18.11. The coordinates "sa_1.pdb" generated by the SA protocol is refined.

```
1  remarks  file nmr/tasa.inp
2  remarks  A protocol for refining NMR structures using a time-averaged
3  remarks  simulated annealing protocol.
4  remarks  author: Sean I. O'Donoghue, 21.9.92
5
6  set seed=7399410 end
7
8  parameter
9          @TOPPAR:parallhdg.pro                     {* Read parameter file. *}
10 end
11
12 structure @g_protein.psf end                       {* Read structure file. *}
13
14 coordinates @sa_1.pdb                             {* Read the coordinates. *}
15
16
17 noe
18 {====>}
19    nres=3000             {*Estimate greater than the actual number of NOEs.*}
20    class all
21 {====>}
```

```
   @g_protein_noe.tbl                          {*Read NOE distance ranges.*}
end

{====>}
@g_protein_dihe.tbl                      {*Read dihedral angle restraints.*}


noe                              {*Parameters for NOE effective energy term.*}
  ceiling=1000
  averaging  * cent
  potential  * square
  scale      * 50.
  sqoffset   * 0.0
  sqconstant * 1.0
  sqexponent * 2
end

parameter                        {*Parameters for the repulsive energy term.*}
    nbonds
      repel=0.75
      rexp=2 irexp=2 rcon=1.
      nbxmod=3
      wmin=0.01
      cutnb=4.5 ctonnb=2.99 ctofnb=3.
      tolerance=0.5
   end
end

restraints dihedral
      scale=5.
end



flags exclude * include bonds angle impr vdw noe cdih end


                                                  {* Initial minimization. *}
minimize powell nstep=300 drop=50 nprint=10 end

               {* Equilibrate the system without time-average constraints. *}

vector do (fbeta = 100.0) (all)                  {* Coupling to heat bath. *}
evaluate  ($bath  = 300.0)                       {* Heat bath temperature. *}

                                                      {* Assign velocities. *}
vector do (vx=maxwell($bath)) (all)
vector do (vy=maxwell($bath)) (all)
vector do (vz=maxwell($bath)) (all)

evaluate ($timestep=0.0005) {ps}
evaluate ($nsteps=25/$timestep)

dynamics  verlet
        nstep=$nsteps
        timestep=$timestep
        iasvel=current
        tcoupling = true
        tbath = 300
        nprint=30 iprfrq = 30
        ntrfrq = 0
end

                                 {* Equilibrate with time-average restraints.*}

noe
        taverage *
                on               {* Turn on time-averaging.                 *}
                tau 1250         {* Set the decay constant in step numbers.  *}
                exponent 3       {* Use r-3 averaging.                      *}
```

```
92                  force nonconservative
93                  reset current
94          end
95          raverage *
96                  on                              {* Turn on running-averaging. *}
97                  exponent 3                      {* Use r-3 averaging.         *}
98                  reset current
99          end
100 end
101
102 evaluate ($timestep=0.001)
103 evaluate ($nsteps=10/$timestep)
104
105 dynamics  verlet
106         nstep=$nsteps
107         timestep=$timestep
108         iasvel=current
109         tcoupling = true
110         tbath = 300
111         nprint=30 iprfrq = 30
112         ntrfrq = 0
113 end
114
115
116 print threshold=0.5 noe
117
118
119 write coordinates output=tasa.pdb  end              {* Write coordinates. *}
120
121 stop
122
123
```

20.11 Ensemble-averaged NOE distance refinement

In structure determination by X-ray crystallography and solution NMR spectroscopy, experimental data are collected as time- and ensemble-averages. Thus, in principle, appropriate time- and ensemble-averaged models should be used. With ensemble-averaging, an ensemble of conformers rather than one single structure is used to satisfy the experimental NMR data. This implies a multifold increase of the number of parameters. A potential danger of this approach is that the resulting improvement of the fit between the model and the data might only reflect the increased number of parameters and thus not be significant. This could result in overfitting the experimental data, especially when the ratio of the number of experimental observables to the number of parameters is less than one. It is therefore important to assess the validity of multi-conformer refinement with ensemble-averaged constraints; to this end, cross-validation is employed (for further reading see Bonvin and Brünger, 1995).

In the examples provided in the tutorial, complete cross-validation is performed on the NOE data as described by Brünger et al. (1993). The NOE-derived distances are partitioned into ten subsets, each of which is, in turn, omitted during refinement. At least ten refinement runs are thus

performed for each particular case (different data or different number of conformers). The rms deviations from the NOE-derived distances and the number of violations exceeding 0.2 Å are monitored for the (omitted) test sets and averaged. No cross-validation is applied to hydrogen bond and dihedral angle restraints.

A requirement for multi-conformer refinement is that specific segid's be defined for all restraints (hydrogen bond-, dihedral angle-, planarity-restraints) except the one on which ensemble-averaging is performed (typically the NOE derived distances). Specific restraints files should thus be created for each number of conformer used in ensemble-averaged refinement.

The first example ("nmr_ensemble/ensemble_cv.inp") is for a monomeric protein. The segid's for the various conformer in the ensemble are set to "C1", "C2", ... The ensemble of conformers is obtained by generating multiple identical copies of an initial structure:

```
    vector do (segid = "C1") ( all )
{====>}                                 {* generates multiple copies        *}
    evaluate ($ncount = 1)
    while ($ncount < $copies) loop GENERATE
        evaluate ($ncount = $ncount + 1)
        evaluate ($chainname = "C" + encode($ncount))
        duplicate
            segid = $chainname
            select = ( segid="C1")
        end
    end loop GENERATE
```

The non-bonded energy terms between conformers are excluded with the following statement:

```
{====>}                                 {* keep molecules from "seeing"     *}
                                        {* each other                       *}
    constraints
        evaluate ($ncount = 0)
        while ($ncount < $copies) loop INTERAC
            evaluate ($ncount = $ncount + 1)
            evaluate ($chainname = "C"+encode($ncount))
            interaction ( segid $chainname ) ( segid $chainname )
        end loop INTERAC
    end
```

Random initial velocities are assigned to each conformer in the ensemble and a simulated annealing protocol consisting of 5 ps constant temperature molecular dynamics at 1000 K with a time step of 0.002 ps is used for refinement, followed by slow-cooling annealing to 1 K with a cooling rate of 50 K / 0.05 ps. Finally, the structures are subjected to 200 steps of conjugated gradient minimization. The setup for ensemble-averaged NOE restraints looks like:

```
    noe
                                        {* turn ON ensemble averaging       *}
        ensemble
            on
        end
    end
```

No DIMER statement is given. This result in the calculation of all NOEs within each monomer (defined by an unique segid) but NOT between monomers.

In case of a multimeric molecule the ENSEmble DIMER statement (see Section (Section 18.1) must be defined to allow intermolecular distance restraints to be taken in the averaging. An example to perform complete cross-validation on the ensemble-averaged NOE distance restraints for a dimeric protein is given in "nmr_ensemble/ensemble_dimer_cv.inp". The segids are set in this case to "A1"/"B1", "A2"/"B2", ... The statement for generating multiple identical copies of the initial structure and excluding non-bonded energy terms between conformers is:

```
    vector do (segid = "A1") ( segid "A")
    vector do (segid = "B1") ( segid "B")
{====>}                                 {* generates multiple copies        *}
    evaluate ($ncount = 1)
    while ($ncount < $copies) loop GENERATE
        evaluate ($ncount = $ncount + 1)
        evaluate ($chainname1 = "A" + encode($ncount))
        evaluate ($chainname2 = "B" + encode($ncount))
        duplicate
            segid = $chainname1
            select = ( segid="A1")
        end
        duplicate
            segid = $chainname2
            select = ( segid="B1")
        end
    end loop GENERATE

{====>}                                 {* keep molecules from "seeing"     *}
                                        {* each other                       *}
    constraints
        evaluate ($ncount = 0)
        while ($ncount < $copies) loop INTERAC
            evaluate ($ncount = $ncount + 1)
            evaluate ($chainname1 = "A"+encode($ncount))
            evaluate ($chainname2 = "B"+encode($ncount))
            interaction ( segid $chainname1 ) ( segid $chainname1 )
            interaction ( segid $chainname1 ) ( segid $chainname2 )
        end loop INTERAC
    end
```

and the NOE ensemble statement becomes:

```
    noe
                                        {* turn ON ensemble averaging       *}
        ensemble
            reset
            on
        end
                                        {* specify allowed intermolecular   *}
                                        {* distances for each dimer within  *}
        evaluate ($ncount = 0)          {* the ensemble                     *}
        while ($ncount < $copies) loop DIM
            evaluate ($ncount = $ncount + 1)
            evaluate ($chainname1 = "A"+encode($ncount))
            evaluate ($chainname2 = "B"+encode($ncount))
            ensemble
               on
               dimer $chainname1 $chainname1
               dimer $chainname1 $chainname2
               dimer $chainname2 $chainname2
```

```
19              end
20          end loop DIM
21      end
```

Cross-validation should be used only to assess the quality of the multi-conformer model whereas all observed data should be used for the final structures using the multi-conformer model giving the best cross-validated measure of the fit. Example scripts for monomeric and dimeric proteins are given in "nmr_ensemble/ensemble.inp" and "nmr_ensemble/ensemble_dimer.inp".

Multi-conformer refinement can result in increased conformational variability so that an average structure, obtained in a standard way by position averaging, might contain unrealistic, strained geometry and give a poor representation of the ensemble. To avoid this problem, DeLano and Brünger (1994) proposed a new method for generating a representative structure from an ensemble based on a three-dimensional probability map (analogous to an electron density map in X-ray crystallography). This method can be used to generate a good representation of an ensemble of multi-conformer structures. Briefly, a probability map is computed from the ensemble. Then, an individual structure is refined against this probability map by Fourier transformation of the map and application of crystallographic refinement techniques in reciprocal space. The same refinement protocol as described by DeLano and Brünger (1994) can be used (200 steps minimization, slow-cooling annealing from 3000 to 300 K over 2.5 ps followed by 1000 steps energy minimization without NMR restraints but with probability map restraints), with, in addition, a slow-cooling annealing from 1000 to 1 K in 5 ps including the NMR restraints. When the ensemble of structures is obtained using multi-conformer ensemble-averaging, the same number of conformers should be used for the probability map refinement. The weight between the empirical energy function and the crystallographic residual are chosen such that the ratio between the gradient norms is approximately two. The crystallographic residual weight is further reduced to one quarter of its initial value in the final refinement with inclusion of the NMR restraints. Two example scripts for a monomeric and a dimeric protein, respectively, are given in the tutorial under "nmr_ensemble/pmrefine.inp" and "nmr_ensemble/pmrefine_dimer.inp".

Other useful scripts for calculating rmsd from an ensemble of multi-conformer structures and splitting the coordinates into single-conformer structures can be found in nmr_ensemble.

20.12 CPU Time Requirements

For the G protein, which is composed of 855 atoms (Gronenborn et al. 1991), the "dg_sub_embed.inp" protocol required 941 sec, the "dgsa.inp" protocol required 9790 sec, the "refine.inp" protocol required 5468 sec, the *ab initio* "sa.inp" protocol required 35932 sec, the "dg_full_embed.inp" protocol required 24324 sec, the "random.inp" protocol required 1846 sec, and the

"generate_template.inp" protocol required 164 sec on a CONVEX C210 for a family of ten structures. The success rate was between 80 and 100% for both the hybrid DG-SA protocols and the *ab initio* SA protocol; the success rate was 70% for the random SA protocol. The default values for the durations of the heating and cooling stages were used.

For the interleukin IL8 protein, which is composed of 2362 atoms (Clore et al. 1990), the "dg_sub_embed.inp" distance geometry protocol required 20741 sec, the "dgsa.inp" protocol required 27274 sec, the "refine.inp" protocol required 14226 sec, the *ab initio* "sa.inp" protocol required 28 hours, the "dg_full_embed.inp" protocol required 8 days, and the "generate_template.inp" protocol required 486 sec on a CONVEX C210 for a family of ten structures. The success rate was between 40 and 50% for the hybrid DG-SA protocols and around 60% for the *ab initio* SA protocol. The default values for duration of heating and cooling stages were used, except for the *ab initio* "sa.inp" protocol, which required 10000 heating steps and 3000 cooling steps.

21 NMR Back-calculation Refinement

This section describes how to back-calculate NOESY spectra from a three-dimensional structure of a molecule, and how to refine an initial model by directly minimizing the difference between the observed 2D NOE intensities and those calculated by the full relaxation matrix approach. Spin diffusion effects are thus accounted for fully during the refinement. Derivatives with respect to atomic coordinates are calculated analytically (Yip and Case 1989). An example of a refinement with this method is Nilges et al. (1991).

For further reading, see Ernst, Bodenhausen, and Wokaun (1987), James et al. (1991), Keepers and James (1984), Lipari and Szabo (1982), Macura and Ernst (1980), Nilges et al. (1991), Solomon (1955), and Yip and Case (1989).

21.1 Setup of the Relaxation Refinement

21.1.1 Syntax

RELAxation { <**relaxation-statement**> } **END** is invoked from the main level of X-PLOR.

<**relaxation-statement**>:==

ASSIgn <**selection**> <**selection**> { < **real** > [< **real** > [< **real** >]]] **[ICALc=**<**real**>] **[IWEIght=**<**real**>] **[REFErence=**<**real**>] **[WORK=**<**real**>] } adds a single cross-peak intensity or a buildup curve between the two specified selected sets of spins. The number of mixing times read depends on the number of CLASs statements issued before the ASSIgn statement. Error estimates can be specified. The format depends on the ERROr_input statement. For examples, see Section 21.7. ICALc, IWEIght, REFErence, WORK are optional statements that describe the calculated data, weight factor, reference and working set flags.

AVERage <**real**> specifies an exponent for calculating the average distance to an UNREsolved group of protons (default: 6).

CALIbrate { <**calibrate-statement**> } **END** calculates the calibration factor between the observed and calculated intensities. Prior to the calibration, this command computes the NOE intensities from the current atomic model unless VALUe is specified.

<**calibrate-statement**>:==

AUTOmatic= ON | OFF sets the calibration factor, if ON, every step during the refinement. At present, this allows the calculation of the gradient only approximately. Therefore, it should be turned OFF during minimization.

GROUp= CLASs | GROUp | ALL for CLASS, calculates the calibration factor separately for each classification; for ALL, one overall calibration factor is used (default: CLASs).

QUALity=<**real**> specifies that cross peaks for which the relative error estimate exceeds the specified value are not used (default: 1).

REFErence= ALL | REFErence for ALL, uses all peaks that meet the criterion set with the EXCLude option for the calculation of the calibration factor; for REFErence, uses only those that were flagged as reference peaks during input (default: ALL).

VALUe class <***real****> sets a value for the calibration factor. Note: if the calibration factor is set by VALUe for a class, it is not recalculated for other classes in the same CALIbrate statement.

CLASsification <**class-name**> partitions an intensity table; it applies to all following ASSIgn entries until another CLASS entry is issued. In contrast to the NOE CLASs statement, the RELAxation CLASs statement can be used to switch between classes during the input of cross-peak intensities. Several class statements can be issued without an intervening ASSIgn statement. This allows the input of buildup curves (default: NIL).

CLGR (CLass_GRoup) <***class****> <**group**> enters a CLASs into a GROUp of spectra (default: none).

CUTOff { <**cutoff-statement**> } **END** applies a cutoff to relaxation matrix and/or gradient calculation. When a cutoff is in effect, for each peak on the data list a separate small relaxation matrix is set up, including only spins within a range of the spin pair specified by the cutoff. The spectrum and the gradient are calculated with this small relaxation matrix.

<**cutoff-statement**>:==

MODE= NONE | GRADient | ALL applies no cutoff, cutoff only for the gradient calculation, or cutoff for both intensity and gradient calculation (default: NONE).

RELAtive_value=<real> specifies a cutoff relative to the actual distance between the two spins in Å (default: 0).

VALUe=<real> specifies the actual value of the cutoff, in Å (default: 6).

EEXPonent <real> is an exponent m of the WELL or PARAbolic energy function (default: 2).

ERROr_input { <error-input-statement> } END defines an error input format for the following ASSIgn statement. It remains in effect until another ERROr_input command is issued.

<errorinput-statement>:==

INPUt=UNIForm | RANGe | PLUSminus specifies the input format (0, 1, or 2 error estimates) for the following ASSIgn statements (default: RANGe): for UNIForm, no error estimate is expected by the program, for RANGe, one error estimate, and for PLUSminus, two, where the first is an error estimate on the minus side, the second on the plus side.

MODE=ABSOlute | RELAtive interprets the error estimate in the ASSIgn statements as absolute or relative (default: ABSOlute).

VALUe=<real> enters the actual value for UNIForm option (default: 0).

GROUp <group-name> defines a group of several CLASses that are recorded with the same physical conditions, like solvent(H_2O, D_2O), field strength, and temperature (default: NIL).

IEXPonent <real> specifies an exponent n of the calculated and observed intensities in the energy function (default: 1).

IWEIght { <iweight-statement> } END applies individual weights w_i to each single term in the energy function. At the moment, the only supported options are weights equal to the inverse of powers of the observed intensity (i.e., $w_i = (\frac{1}{I_i^o})^g/2$). If the exponent g matches the exponent m of the energy function, the weighting is equivalent to refining relative intensity differences. The minimum weight applied is always 1.

<iweight-statement>:==

CUTOff=<real> is the maximum weight applied (default: 10000).

EXPOnent=<real> specifies an exponent g of the observed intensity in the weight function $w_i = (\frac{1}{I_i^o})^g/2$ (default: 2).

QUALity=<real> stipulates that cross peaks for which the relative error estimate exceeds the specified value are not used in determining the maximum weight (default: 1).

MINIntensity <*class*> <real> sets the minimum observable intensity for each class. If either the observed or the calculated intensity drop below this value, this intensity is excluded from the calculation of the target function (Eq. 21.9) (default: smallest number used by X-PLOR).

NREStraints=<integer> is the required parameter that specifies the maximum expected number of intensities. It has to be greater than or equal to the actual number. The parameter is used to assign space dynamically for the intensity list (default: 200).

OCCUpancy <group> <selection> <real> sets the occupancy for selected spins (for example, 0.9 for exchanging amid protons) (default: 1.0).

OMEGa <*group*> <real> specifies the spectrometer frequency, in Hz (default: $500*10^6$).

POTEntial=WELL | PARAbolic provides two alternatives: WELL uses a flat-bottom potential with bounds at error estimates, PARAbolic a (generalized) parabola with exponent EEXP and center at the specified intensity (default: WELL).

PREDict_intensities <class> { <predict-intensities-statement> } END calculates a spectrum from coordinates with the parameters valid for the specified class. Note that this destroys any existing experimental intensity database. The spectrum can be calculated for parts of a molecular structure.

<PREDict_intensities-statement>:==

CUTOFf=<real> specifies a distance cutoff for cross-peak calculation (default: 10).

CUTON=<real> specifies a distance cuton for cross-peak calculation (default: 0).

DIAGonal=ON | OFF prints diagonal peaks (default: OFF).

FORMat=NORMal | LIST is an output format. NORMal produces a listing of the predicted intensities, the distances between the spins, and the scale factors. LIST produces a listing of the predicted intensites in the format of an intensity INPUT file (default: NORMal).

FROM=<selection> selects the first part of a molecular structure (default: all NMR-active atoms).

THREshold=<real> specifies the minimum intensity that is printed (default: 0).

TO=<selection> selects the second part of a molecular structure (default: all NMR-active atoms).

PRINt_deviations { <**print-deviation-statement**> } **END** prints an NMR R value and produces a listing of the observed and computed intensites for differences larger than a specified threshold. It uses the functional form of the energy function (WELL or PARAbolic and the value of IEXP) specified before issuing the statement.

<**print-deviation-statement**>:==

FORMat=NORMal | LIST is an output format. NORMal produces a listing of the calculated and observed intensities and their differences. LIST produces a listing of the calculated intensities in the format of an intensity input file.

SELEct=<selection> calculates the R value with only a subset of the atoms. This allows the calculation of local R values.

THREshold=<real> specifies the minimum deviation that is printed, in the arbitrary units of the input intensities. For a complete list of deviations (including 0), THREshold should be set to a negative number (default: 0).

FSET=WORK | TEST select set to print from (only for FORMat=LIST). (default: WORK).

ICALc=<logical> prints calculated intensity (only for FORMat=LIST) (default: FALSE).

SCALe=<logical> if true, calculated data will be scaled (default: FALSE).

IOBS=<logical> prints observed intensity and error estimates (only for FORMat=LIST) (default: TRUE).

IWEIght=<logical> prints weight (only for FORMat=LIST) (default: FALSE).

REFErence=<logical> prints reference flag (only for FORMat=LIST) (default: FALSE).

WORK=<logical> prints working set flags (only for FORMat=LIST) (default: FALSE).

REFErence flags intensities read by the following ASSIgn statements as reference peaks for the calculation of the calibration factor (see CALIbrate statement), and remains in effect until another CLASs statement is issued. This flag allows one to use a subset of peaks for the calibration, e.g., peaks that do not depend very much on the conformation of the molecule. The reference peaks are written into a separate file, and the reference flag is set before reading that file.

RESEt erases the current relaxation database.

SELEct <**group**> <**selection**> selects the "NMR-active" atoms (default: (HYDRogen)). At present, only protons can be used in the refinement.

The NMR parameters for protons are applied to all selected atoms. This command initializes the array that stores information about unresolved groups (see below). If the command is not issued at the beginning of the relaxation matrix setup, all hydrogens are used by default. Since a hydrogen is defined by its mass in X-PLOR, care should be taken in this case to modify the masses of hydrogens for simulated annealing only after the relaxation matrix parameters have been set up. This statement resets the parameters for unresolved intensities, i.e., the UNREsolved statement needs to be repeated after the SELEction statement.

SORT_intensities sorts the intensity list so that peaks with different mixing times belonging to the same spin pair are consecutive on the list. If a cutoff is used, a new relaxation matrix generally has to be set up and diagonalized for each peak on the data list. However, if the only difference between two entries on the data list is the mixing time, the relaxation matrix does not have to be recalculated and diagonalized. Sorting can thus save a considerable amount of CPU time.

TAUCorrelation <**group**> { <**TAUC-statement**> } **END** specifies the correlation time and the order parameter, depending on the chosen model.

<**TAUCorrelation-statement**>:==

MODEl= RIGId | LIPAri provides two alternatives: RIGId expects no order parameter; LIPAri expects one overall correlation time and one order parameter. This applies to both the ISOTropic and the VECTor case.

ISOTropic < **real** > [< **real** >] specifies an isotropic correlation time and an order parameter (default: 10 ns, 1).

RESEt initializes a table.

VECTor <**selection**> <**selection**> < **real** > [< **real** >] specifies a correlation time (in seconds) and an order parameter for a specified vector.

TAUMix <***class***> <**real**> specifies the mixing time τ_m, in seconds (default: 0.1 sec).

TOLErance=<**real**> if not equal to zero, approximates the exact calculation of the $E_{relaxation}$ energy function and its first derivatives with respect to the atomic coordinates by freezing $E_{relaxation}$ until atoms have moved by less than TOLErance Å from the point at which $E_{relaxation}$ was last computed. This is particularly useful for molecular dynamics calculations. It should be set to 0 during minimization to avoid inconsistencies in the force field (default: 0).

UPDate computes the NOE intensities from the current atomic model.

UNREsolved <**group**> { <**unresolved-statement**> } **END** replaces protons with unresolved chemical shifts by a single spin. The distance to an unresolved group is calculated as $< r^{-av} >^{1/av}$, where the exponent av is set with the AVERage command. A diagonal leakage rate is added to the relaxation matrix for each unresolved group. The main use of this command is for methyl groups.

<**unresolved-statement**>:==

1-2 treats each methylene group as an unresolved group of protons. A methylene group is recognized by X-PLOR as two hydrogens bound to a heavy atom.

METHyl treats each methyl group as an unresolved group of protons. A methyl group is recognized by X-PLOR as three hydrogens bound to a heavy atom.

NONE deletes all previously defined unresolved groups.

SELECT=<**selection**> treats the selected atoms as an unresolved group of protons.

WEIGht <***class***> <**real**> specifies the energy constant W_N (overall weight) (default: 1).

ZLEAkage_rate <***group***> <**real**> specifies the overall diagonal leakage rate R_{leak}, in $\sec^{-1}$ (default: 0).

21.1.2 Requirements

The molecular structure must be defined, and the main coordinate set must be well defined.

21.1.3 Output

The symbol $RNMR is set to the generalized NMR R value after each evaluation of the $E_{relaxation}$ energy term (Eq. 21.9).

21.2 The Relaxation Matrix

The basis for the refinement is the calculation of the volume of a cross peak between spins i and j, I_{ij}^c, from the atomic coordinates by means of the relaxation matrix $\mathbf{R}$ (Macura and Ernst 1980; Ernst et al. 1987; Keepers and James 1984):

$$I_{ij}^c \propto [\exp(-\mathbf{R}\tau_m)]_{ij}, \tag{21.1}$$

where τ_m is the mixing time. The relaxation matrix $\mathbf{R}$ is a function of the transition rates Ω^{ij}

$$\mathbf{R}_{ij} = \begin{cases} \Omega_2^{ij} - \Omega_0^{ij} & \text{if } i \neq j \\ \sum_{k \neq j} \Omega_0^{kj} + 2\Omega_1^{kj} + \Omega_2^{kj} + R_{leak} & \text{if } i = j \end{cases} \tag{21.2}$$

which are determined by spectral densities and dipolar coupling strengths (Solomon 1955):

$$\begin{aligned} \Omega_0^{ij} &= d_{ij} J(0) \\ \Omega_1^{ij} &= \tfrac{3}{2} d_{ij} J(\omega) \\ \Omega_2^{ij} &= 6 d_{ij} J(2\omega) \end{aligned} \tag{21.3}$$

and

$$d_{ij} = \frac{\gamma^4 \hbar^2}{10 r_{ij}^6} \tag{21.4}$$

γ is the gyromagnetic ratio of the proton and r_{ij} the distance between spins i and j. At present, only protons can be used in the refinement. R_{leak} describes the non-NOE magnetization losses from the lattice.

In the simplest model, it is assumed that a single isotropic correlation time τ_c is sufficient to describe the spectral densities $J(\omega)$ (Solomon 1955):

$$J(\omega) = \frac{\tau_c}{1 + \omega^2 \tau_c^2}. \tag{21.5}$$

A step beyond this simple model is the "model-free" approach of Lipari and Szabo (1982), where the internal motion is described by two parameters, an effective correlation time τ_e and an order parameter S^2:

$$J(\omega) = S^2 \frac{\tau_c}{1 + \omega^2 \tau_c^2} + (1 - S^2) \frac{\tau_e}{1 + \omega^2 \tau_e^2} \tag{21.6}$$

X-PLOR uses an approximation of this equation that assumes that the internal motion is much faster than the overall rotation of the molecule (i.e., $\tau_e \ll \tau_c$), such that the second term in the equation becomes negligible. In order to take into account the different motional behavior of different parts of the molecule, different correlation time and order parameters can be entered for different proton-proton vectors.

Groups of protons whose resonances are degenerate due to motion (in general, mostly methyl groups) are treated roughly as in CORMA, version 1.5 (Keepers and James 1984). (Note that cross peaks which are ambiguous due to overlap should be dealt with in a different way; see the example input file in Section 21.7.) Each such group is represented by one spin, whose intensity is scaled by the number of protons in the group, and the distance to the group is calculated as the $< r^{-3} >^{-1/3}$ or $< r^{-6} >^{-1/6}$ average over the protons in the group (Eq. 21.4). In addition, a diagonal leakage rate is added for each group of protons.

Protons can be removed from the spin system (exchangeable protons in D_2O spectra, deuterium-labeled molecules), or their appropriate occupancy can be specified (exchangeable protons in H_2O spectra).

21.3 Analytical Expression for the Gradient

The derivative of I_{ij}^c with respect to a coordinate μ (Eq. 12 of Yip and Case 1989) can be written as

$$\nabla_\mu I_{ij}^c = \nabla_\mu [\exp(-\mathbf{R}\tau_m)]^{(ij)} = \mathrm{Trace}[(\nabla_\mu \mathbf{R})\mathbf{L}\mathbf{F}^{(ij)}\mathbf{L}^T] \tag{21.7}$$

where $\mathbf{F}^{(ij)}$ is defined as

$$\mathbf{F}_{ru}^{(ij)} \equiv \begin{cases} -\mathbf{L}_{ir}\mathbf{L}_{uj}^T \dfrac{\exp(-\lambda_r\tau) - \exp(-\lambda_u\tau)}{\tau(\lambda_r - \lambda_u)} & \text{if } r \neq u \\ \mathbf{L}_{ir}\mathbf{L}_{rj}^T \exp(-\lambda_r\tau) & \text{else} \end{cases} \tag{21.8}$$

$\mathbf{L}$ and $\mathbf{\Lambda}$ are the matrix of eigenvectors and eigenvalues of $\mathbf{R}$, respectively. λ_r is the rth eigenvalue of the relaxation matrix.

21.4 The $E_{relaxation}$ Energy Term

Once the NOESY spectrum and its gradient are calculated, the relaxation energy $E_{relaxation}$ can be expressed as a function of the difference between (functions of) observed and calculated intensities, and analytic derivatives with respect to atomic coordinates can be readily obtained by using the chain rule

$$E_{relaxation} = W_N \sum_{Spectra} \sum_{i=1}^{N_S} w_i \cdot \mathrm{well}(I_i^c, k_S I_i^o, \Delta_i, n)^m \tag{21.9}$$

where W_N is the energy constant for the relaxation term, I_i^c and I_i^o are respectively the calculated and observed intensities, Δ_i is an error estimate for I_i^o, w_i is a weight factor, k_S is the calibration factor for each spectrum, and N_S is the number of cross peaks in each spectrum. The function $\mathrm{well}(a, b, \Delta, n)$ is defined as the absolute value of the difference between the nth powers of a and b, where b has an error estimate Δ:

$$\mathrm{well}(a, b, \Delta, n) \equiv \begin{cases} (b - \Delta)^n - a^n & \text{if } a^n < (b - \Delta)^n \\ 0 & \text{if } (b - \Delta)^n < a^n < (b + \Delta)^n \\ a^n - (b + \Delta)^n & \text{if } a^n > (b + \Delta)^n \end{cases} \tag{21.10}$$

The individual error estimates Δ_i reflect the errors in the peak volumes, usually subjective estimates, especially due to noise and spectral overlap. At present, the error estimates are also used to ascertain if a measured intensity is to be used in the determination of the overall calibration factor.

Values for the exponents of $n = 1/2$ and $m = 2$ (Eqs. 21.9 and 21.10) correspond to the refinement of the residual in X-ray crystallography. These values tend to put a high weight on the large intensities, resulting in a bad fit of intensities for which the calculated value is too small. Following a suggestion by James et al. (1991), use $m = 2$ and $n = \frac{1}{6}$. A value of $m = 1$

results in the refinement of the R value directly, instead of the residual. The discontinuity of the gradient may lead to instabilities during the refinement.

In addition to the overall weight W_N, individual weights w_i can be applied to each term in the sum in Eqs. 21.9 and 21.12, e.g., in order to increase the relative weight of the small intensities. (It should be noted that this is achieved already by setting $n = \frac{1}{6}$ in Eq. 21.9.) The scheme $w_i = (\frac{1}{I_i^o})^{m/2}$ corresponds to a common weighting scheme used in crystallography if experimental σ values are unreliable or unavailable. In the NMR case, however, there is no theoretical justification for this weighting scheme. (In crystallography, the statistical error σ of an intensity measurement I_i^o is $\sqrt{I_i^o}$.) The weights are scaled such that $\min(w_i) = 1$.

The calibration factor between observed and back-calculated intensities is determined simply as the ratio of the sums of all calculated and observed intensities:

$$k_S = \frac{\sum_{i=1}^{N'_S} (I_i^c)^n}{\sum_{i=1}^{N'_S} (I_i^o)^n} \tag{21.11}$$

This ratio can be determined separately for each spectrum, or overall for all data points. Volumes that are not very reliable (i.e., they have a large error estimate) can be excluded from the ratio. The calibration factor can be updated automatically at every step. For technical reasons, this does not allow the calculation of the exact derivatives at present, so the calibration should not be updated automatically at every step during conjugate gradient minimization. During annealing, the effects due to the error in the gradient are negligible.

21.5 Cutoffs

The direct refinement against NOESY intensities is more CPU intensive than distance-restrained refinement. Several means have been implemented to reduce the CPU time requirements.

The tolerance statement specifies the maximum distance that an NMR-active atom is allowed to move until the relaxation matrix pseudoenergy is recalculated. If the tolerance is larger than 0, the pseudoenergy is not calculated at every step. The optimal value of the tolerance depends on the energy parameters, weights on the relaxation matrix pseudoenergy, masses of the atoms, and annealing temperature. Too large a value for the tolerance (i.e., the relaxation matrix pseudoenergy is recalculated too rarely) heats the system and induces large temperature fluctuations, and the molecule may be trapped in the starting conformation. A value of 0.05 Å is a good starting point.

The largest reduction in computation time is achieved by the introduction of a distance cutoff for the relaxation matrix and gradient calculations. To a good approximation, the size of a 2D NOE cross peak between spins i and

j is affected only by the relaxation pathways via spins close to i or j. Thus, individual distance cutoff spheres are used around i and j for the calculation of I_{ij}^c and the contribution to the gradient due to this cross peak. For every pair of spins, a relaxation matrix is generated and diagonalized separately.

The cutoff determines the number of cross relaxation pathways that are included in the calculation. Its optimal value depends on the longest mixing time and the rotational correlation time of the refined molecule. For 200 ms and 2.3 ns, a value of 4.5 Å is sufficient. The cutoff can also be specified relative to the actual distance between the protons i and j. In this case, the distance between the two protons is measured, and the cutoff is set to the distance times the specified factor.

21.6 Assessing the Quality of the Final Structure

As a measure of the fit of the refined coordinates to the NOESY data, a generalized R value is used:

$$R^n = \frac{\sum_{Spectra} \sum_{i=1}^{N_S} w_i \cdot \mathrm{well}(I_i^c, k_S I_i^o, \Delta_i, n)}{\sum_{Spectra} \sum_{i=1}^{N_S} w_i \cdot (k_S I_i^o)^n} \tag{21.12}$$

For $n = 1/2$, $\Delta_i = 0$, and $w_i = 1$, R^n corresponds to the R value commonly used in crystallography (Stout and Jensen 1989). For $n = \frac{1}{6}$, R^n corresponds to the value suggested by James et al. (1991).

21.7 Input of the Experimental Data

The experimental data can be classified in two ways. Classes of restraints usually correspond to a single spectrum (mixing time). A group of classes corresponds to a group of spectra measured under the same physical conditions (relaxation network, solvent, magnetic field strength, temperature). This distinction is made since, in principle, a change in the physical conditions makes it necessary to recalculate the relaxation matrix, while a change in the mixing time does not. Order parameters, correlation times, occupancy values, and the relaxation network can thus be specified for each group of classes separately.

The intensity data are read into a class, and then the group that the class belongs to is specified. The syntax is similar to the NOE assign statement (see Section 18.1), but the input format is more flexible, to allow the user to input NOE buildup curves and error estimates. Before issuing an assign statement, one has to specify how many mixing times are to be read in at a time and what format for the error estimates the program should expect. The number of classification statements determines how many mixing times are included in each assign statement. The error_input statement determines the format of the error estimates. Ambiguous NOESY cross peaks can be

entered using multiple atom selections. During the refinement, the sum of the corresponding calculated intensities is restrained to the sum of the observed ones.

The example intensity file below shows a number of different options. Note that the class and error statements remain valid until other class or error statements are entered.

```
remarks file: nmr_relaxation/sample.tbl

{* ------------------------------------------------------------------------  *}
{* Four mixing times.                                                        *}
{* Volume/error input as                                                     *}
{* value    error on minus side    error on plus side,                      *}
{* so four triplets of numbers are read.                                     *}


class  50
class  100
class  150
class  200
error_input
    input=plusminus  mode=absolute
end
assign (resid  12 and name hn)(resid 11 and name hn)
    29.7 0.3 0.3   44.0 0.3 0.3   47.7 0.3 0.3   46.7 0.3 0.3
assign...

{* ------------------------------------------------------------------------  *}
{* Ambiguous cross peak:                                                     *}
{* the sum of all peaks possible from the selection is compared.             *}

assign (resid 7 and name hn)
       ((resid 6 and name ha) or (resid 7 and name ha))
    2.7 0.3 0.3    8.5 0.5 0.5    12.6 0.3 0.3    14.5 0.3 0.3

{* ------------------------------------------------------------------------  *}
{* Methyl group (hb#) or ambiguous cross peak:                               *}
{* if the hb# is defined as methyl group (with UNREsolved ... METHyl END), *}
{*    one peak volume will be calculated;                                    *}
{* if the beta protons are treated as separate spins, a volume will be      *}
{*    calculated for every possible pair and summed up.                      *}

assign (resid 7 and name hn) (resid 8 and name hb#)
    2.7 0.3 0.3    8.5 0.5 0.5    12.6 0.3 0.3    14.5 0.3 0.3

{* ------------------------------------------------------------------------  *}
{* One mixing time, volume/error input as                                    *}
{*   central value      range.                                               *}

class  150
error_input
    input=range  mode=absolute
end
assign (resid 3 and name  hn) (resid 2 and name hn)   2.99 0.3

{* ------------------------------------------------------------------------  *}
{* Four mixing times, a uniform error of 1.0 is assumed.                     *}

class  50
class  100
class  150
class  200
error_input
    input=uniform  mode=absolute  value=1.0
end
assign (resid 3 and name hn)(resid 18 and name hb#) 1.2 2.8 5.0 4.5

```

```
{* ------------------------------------------------------------------  *}
{* Four mixing times, uniform error value, relative error (10 %).     *}

class  50
class  100
class  150
class  200
error_input
    input=uniform  mode=relative  value=0.1
end
assign (resid 3 and name hn)(resid 18 and name hb#) 3.0 6.0 8.0 9.0
```

21.8 Prediction of a NOESY Spectrum

The values of the order parameters in the example file below were calculated from a 5 nsec molecular dynamics trajectory of BPTI in vacuum (Schneider, Nilges, and Brünger, in preparation). The correlation time is the result of the grid search that is described.

```
remarks file: nmr_relaxation/spectrum.inp
remarks Prediction of a spectrum from a 3D structure

structure @c2.psf end                                  {*Read structure file.*}
coor @c2_01_ref.pdb                                      {*Read coordinates.*}

relaxation

{*========================== Specify the physical parameters of the system.*}

    group      DMSO                {*A group of spectra, DMSO, is defined.*}
    unresolved DMSO
        methyl                              {*Methyl protons are unresolved.*}
    end
    average 3                                {*R-3 average to methyl groups.*}
    omega      DMSO 500.0e6                 {*Spectrometer frequency, in Hz.*}
    taucorrel  DMSO                               {*Correlation times, in s.*}
        model lipari                                     {*Order parameters.*}
        vector (name hn or name ha)
               (name hn or name ha)                      0.75e-9 0.85
        vector (name hn or name ha)
               (name h* and not (name hn or name ha))    0.75e-9 0.80
        vector (name h* and not (name hn or name ha))
               (name h* and not (name hn or name ha))    0.75e-9 0.65
    end

    classification D300                  {*Define a classification, D300,*}

    clgroup     D300 DMSO                {*which belongs to group DMSO.  *}
    taumix      D300 0.30                              {*Mixing time, in s.*}

    cutoff             {*No cutoff is used in setting up relaxation matrix.*}
        mode none
    end

    calibrate            {*A calibration factor can be entered for scaling.*}
        value D300 0.1085E-01
    end

{*========================== Predict spectrum with the parameters for D300.*}

    set print c2_01_ref.spect    end{*Open the output file for the spectrum.*}

    predict D300
        from (resid 3) to (resid 4)             {*Select part of structure.*}
```

```
46          threshold 0.10                          {*Minimum intensity printed.*}
47          cutoff 6.0                                  {*Distance cutoff, in A.*}
48      end
49
50  end
51
52  stop
```

The spectrum file contains the identifications of the two spins involved in each cross peak, the cross peak volume, the distance between the two spins, and the scale factor set by the calibration statement. Note that for the methyl groups, all involved protons are listed.

```
1   Intensities for class D300 in spectra-group DMSO
2         I-ATOMS              J-ATOMS          intensity    distance  scale-factor
3
4         3    THR  HN         4    LYS  HN    0.2745E+00  0.3615E+01  0.1085E-01
5
6         3    THR  HN         4    LYS  HD1   0.1301E+00  0.4003E+01  0.1085E-01
7
8         3    THR  HN         4    LYS  HD2   0.3219E+00  0.3212E+01  0.1085E-01
9
10        3    THR  HN         4    LYS  HZ1   0.1276E+01  0.3044E+01  0.1085E-01
11                             4    LYS  HZ3
12                             4    LYS  HZ2
13
14        3    THR  HA         4    LYS  HN    0.2371E+01  0.2346E+01  0.1085E-01
15
16        3    THR  HA         4    LYS  HD2   0.3193E+00  0.3348E+01  0.1085E-01
17
18        3    THR  HA         4    LYS  HZ1   0.1137E+01  0.3214E+01  0.1085E-01
19                             4    LYS  HZ3
20                             4    LYS  HZ2
21
22        3    THR  HB         4    LYS  HN    0.1357E+00  0.4159E+01  0.1085E-01
23
24        3    THR  HG1        4    LYS  HB2   0.2758E+00  0.3318E+01  0.1085E-01
25
26        3    THR  HG1        4    LYS  HD2   0.1628E+00  0.3562E+01  0.1085E-01
27
28        3    THR  HG21       4    LYS  HZ1   0.1338E+00  0.5633E+01  0.1085E-01
29        3    THR  HG23       4    LYS  HZ3
30        3    THR  HG22       4    LYS  HZ2
31
```

21.9 Refinement against NOESY Intensities

The structure of a protein is refined against the NOESY intensities by a combination of simulated annealing and conjugate gradient minimization. The values of the order parameters in the example file below were calculated from a 5-nsec molecular dynamics trajectory of BPTI in vacuum (Schneider, Nilges, and Brünger, in preparation), and they can serve as a starting point for further refinement by a grid search.

The file below is an example of possible refinements by minimization and/or simulated annealing.

```
1  remarks file: nmr_relaxation/refine_new.inp
2  remarks Relaxation matrix refinement using the method of
```

```
remarks  Yip and Case (1989)

set seed 584930 end                  {*Seed for random velocity assignment.*}

structure @c2.psf end                                  {*Read structure file.*}
coor @c2_01_sa.pdb                                        {*Read coordinates.*}

parameter                                             {*Set energy parameters.*}
    @TOPPAR:parallhdg.pro
        nbonds
        atom  nbxmod -3  tolerance 0.5
        repel 0.75  rcon 4.0  rexp 2  cutnb 4.5
    end
end
flags include relaxation end          {*Include the relaxation energy term.*}

relaxation
    nrestraints 200                        {*Expected number of restraints.*}

{*========================== Specify the physical parameters of the system.*}

    group      DMSO                 {*A group of spectra, DMSO, is defined.*}
    unresolved DMSO
        methyl                             {*Methyl protons are unresolved.*}
    end
    average 3                               {*R-3 average to methyl groups.*}
    omega      DMSO 500.0e6                {*Spectrometer frequency, in Hz.*}
    taucorrel  DMSO                              {*Correlation times, in s.*}
        model lipari                                    {*Order parameters.*}
        vector (name hn or name ha)
               (name hn or name ha)                     0.75e-9 0.85
        vector (name hn or name ha)
               (name h* and not (name hn or name ha))   0.75e-9 0.80
        vector (name h* and not (name hn or name ha))
               (name h* and not (name hn or name ha))   0.75e-9 0.65
    end

    classification D300                    {*Define a classification, D300,*}
    @c2_300.int

    clgroup     D300 DMSO                  {*which belongs to group DMSO.  *}
    taumix      D300 0.30                             {*Mixing time, in s.*}
    minint      D300 0.05                  {*Minimum measurable intensity.*}

    cutoff               {*Cutoff used for gradient and relaxation matrix.*}
        mode all  value 4.5
    end

    potential parabola                     {*Error estimates are not used.*}
    iexp 0.1666666667                    {*Dev = I_calc^iexp - I_obs^iexp.*}
    eexp 2                                                {*E = Dev^eexp.*}

    calibrate         {*Calculate spectrum and turn automatic rescaling on.*}
        quality 0.33  automatic on  reference all  group class
    end

    print  threshold 9999  end        {*Print R value of initial structure.*}
end

{*============================================== Cool from 1000 K to 100 K.*}

vector do (mass = 10) (all)                              {*Uniform masses.*}
vector do (fbeta= 50) (all)   {*Friction coefficient for heatbath coupling.*}

evaluate ($init_t = 1500)            {*Initial annealing temperature, in K.*}
evaluate ($final_t =  75)              {*Final annealing temperature, in K.*}
evaluate ($tempstep = 25)               {*Temperature step per cycle, in K.*}

vector do (vx = maxwell($init_t)) (all)        {*Assign initial velocities.*}
vector do (vy = maxwell($init_t)) (all)
```

```
vector do (vz = maxwell($init_t)) (all)

evaluate ($totalstep = 2000)              {*Total annealing time, in steps.*}
evaluate ($nstep = int($totalstep*$tempstep/($init_t-$final_t)))

relaxation
    weight    D300  1000                        {*Set the relaxation weight.*}
    tolerance 0.0             {*Calculate relaxation energy at every step.*}
end

evaluate ($bath = $init_t)
while ($bath > $final_t) loop temp
    dynamics  verlet
        nstep=$nstep timest=0.001 {ps}
        iasvel=current
        tcoupling=true  tbath=$bath
        nprint=$nstep iprfrq=$nstep
    end
    evaluate ($bath = $bath - $tempstep)     {*Reduce heatbath temperature.*}
end loop temp

{*======================================== Conjugate gradient minimization.*}

evaluate ($count = 0)
while ($count < 3) loop mini
    evaluate ($count = $count + 1)
    relaxation
        calibrate                    {*Automatic calibration has to be off.*}
           quality 0.33  automatic off  reference all  group class
        end
    end
    minimize powell nstep 30 end
end loop mini

write coor output= refine.pdb end

stop
```

21.10 Simultaneous Refinement with H_2O and D_2O Spectra

The example file below shows how to set up the relaxation matrix calculations when data from different solvents are used. For the D_2O spectrum, only nonexchangeable protons are selected; for the H_2O spectrum, the occupancy of the exchangeable protons is set to 0.9.

```
remarks file: nmr_relaxation/multiplesolvent.inp

vector ident (store1)                           {* Non-exchangable protons.*}
                   (name h* and not (name hn or name ht3
                or (resn lys and name hz#)
                or (resn arg and (name he or name hh#))
                or (resn asn and name hd2#)
                or (resn gln and name he2#)
                or (resn ser and name hg)
                or (resn thr and name hg1)
                or (resn tyr and name hh)
                or (resn trp and name he1)
                or (resn his and name hd1)))

vector ident (store2)                             {* Exchangable protons. *}
                   (name hn or name ht3
                or (resn lys and name hz#)
```

```
                or (resn arg and (name he or name hh#))
                or (resn asn and name hd2#)
                or (resn gln and name he2#)
                or (resn ser and name hg)
                or (resn thr and name hg1)
                or (resn tyr and name hh)
                or (resn trp and name he1)
                or (resn his and name hd1))


relaxation

    nrestraints 200                        {*Expected number of restraints.*}

{*========================== Specify the physical parameters of the system.*}

    group      H2O                   {*A group of spectra, DMSO, is defined.*}
    select     H2O  (name h*)                          {*All protons relax.*}
    unresolved H2O
        methyl                               {*Methyl protons are unresolved.*}
    end
    average 3                                 {*R-3 average to methyl groups.*}
    omega      H2O  500.0e6                   {*Spectrometer frequency in Hz.*}
    taucorrel  H2O                            {*Correlation times in seconds.*}
        model lipari                                      {*Order parameters.*}
        vector (name hn or name ha)
               (name hn or name ha)                        0.75e-9 0.85
        vector (name hn or name ha)
               (name h* and not (name hn or name ha))   0.75e-9 0.80
        vector (name h* and not (name hn or name ha))
               (name h* and not (name hn or name ha))   0.75e-9 0.65
    end
    occupancy H2O   (store2)  0.9


    group      D2O                {* A group of spectra, D2O, is defined. *}
    select     D2O  (store1)
    unresolved D2O
        methyl                            {* Methyl protons are unresolved. *}
    end
    omega      D2O  500.0e6               {* Spectrometer frequency in Hz. *}
    taucorrel  D2O                            {* Correlation times ( in s) *}
        model lipari                          {* and order parameters.     *}
        vector (store1 and (name hn or name ha))
               (store1 and (name hn or name ha))       0.75e-9 0.85
        vector (store1 and (name hn or name ha))
               (store1 and not (name hn or name ha))   0.75e-9 0.80
        vector (store1 and not (name hn or name ha))
               (store1 and not (name hn or name ha))   0.75e-9 0.65
    end

    average 3                                 {* r-3 average to methyl groups *}

    classification H300                       {*Define a classification, D300.*}
    {===>} @h300.tbl                          {*Read NOESY volume file which  *}
    clgroup     H300 H2O                      {*belongs to group H2O.         *}
    taumix      H300 0.30                     {*Mixing time, in seconds.      *}
    minint      H300 0.05                     {*Minimum measurable intensity. *}

    classification D200                       {*Define a classification, D200.*}
    {===>} @d200.tbl                          {*Read NOESY volume file which  *}
    clgroup     D200 D2O                      {*belongs to group D2O.         *}
    taumix      D200 0.20                     {*Mixing time, in seconds.      *}
    minint      D200 0.05                     {*Minimum measurable intensity. *}

end
```

21.11 Calculation of Different R Values

R values are output separately as single numbers for each classification and separately in intensity shells for each group of spectra. The overall R value is stored in the variable $RNMR.

```
remarks file: nmr_relaxation/rfactor.inp

structure @c2.psf end                                  {*Read structure file.*}
coor @c2_01_ref.pdb                                      {*Read coordinates.*}

relaxation
    nrestraints 200                          {*Expected number of restraints.*}

{*========================== Specify the physical parameters of the system.*}

    group      DMSO                  {*A group of spectra, DMSO, is defined.*}
    unresolved DMSO
        methyl                               {*Methyl protons are unresolved.*}
    end
    average 3                                 {*R-3 average to methyl groups.*}
    omega      DMSO 500.0e6                  {*Spectrometer frequency, in Hz.*}
    taucorrel  DMSO                               {*Correlation times, in s.*}
        model lipari                                      {*Order parameters.*}
        vector (name hn or name ha)
               (name hn or name ha)                       0.75e-9 0.85
        vector (name hn or name ha)
               (name h* and not (name hn or name ha))   0.75e-9 0.80
        vector (name h* and not (name hn or name ha))
               (name h* and not (name hn or name ha))   0.75e-9 0.65
    end

    classification D300                     {*Define a classification, D300,*}
    @c2_300.int

    clgroup     D300 DMSO                   {*which belongs to group DMSO.  *}
    taumix      D300 0.30                                {*Mixing time, in s.*}
    minint      D300 0.05                  {*Minimum measurable intensity.*}

    cutoff
        mode all  value 4.5
    end

{*=========================================  Calculation of R-6 R value.*}

    set print c2_r6.disp end             {*Open output file for deviations.*}

    potential parabola                     {*Error estimates are not used.*}
    iexp 0.1666666667                    {*Dev = I_calc^iexp - I_obs^iexp.*}

    calibrate
        quality 0.33  automatic off  reference all  group class
    end

    print threshold -0.1 end   {*List all deviations and calculate R value.*}

{*=========================================  Calculation of X-ray R value.*}

    set print c2_X.disp end              {*Open output file for deviations.*}

    potential parabola                     {*Error estimates are not used.*}
    iexp 0.1666666667                    {*Dev = I_calc^iexp - I_obs^iexp.*}

    calibrate
        quality 0.33  automatic off  reference all  group class
    end

    print threshold -0.1 end        {*List all peaks and calculate R value.*}
```

```
end

stop
```

The following example shows a part of the output file, i.e., the calculated calibration factor and the R value. The intensities are calculated from the coordinates by the calibration statement, and the optimal calibration factor is printed for each classification separately. The R value is printed first for each classification and then for each group broken down in intensity shells.

```
 RELAx>    calibrate
 CALIbrate>        quality 0.33  automatic off  reference all  group class
 CALIbrate>    end
 RELAX: intensities updated for calibration

             class   N(reference)   N(all)  calibration
            ------------------------------------------
              D300              0        23   .5220E+00
            ------------------------------------------
 RELAx>
 RELAx>    print threshold -0.1 end      {List all deviations and calculate R.}
 Class D300: R-factor:    .6594E-01 R.m.s. difference:     .8038E-01
 RELAX: overall R-factor statistics
            ------------------------------------------------------
              min. int.   max. int.    N   R-factor   r.m.s. diff
            ------------------------------------------------------
              .2550E+01   .5100E+01    1   .4804E-01   .6302E-01
              .1275E+01   .2550E+01    4   .5346E-01   .8720E-01
              .6375E+00   .1275E+01    4   .4047E-01   .4735E-01
              .3187E+00   .6375E+00    9   .6331E-01   .7320E-01
              .1594E+00   .3187E+00    5   .1292E+00   .1204E+00
              .7969E-01   .1594E+00    4   .5022E-01   .5105E-01
            ------------------------------------------------------
              .0000E+00   .5100E+01   27   .6594E-01   .8038E-01
            ------------------------------------------------------
```

A local R value can be calculated using the select option in the print statement. The following loop calculates the R value for each residue in the sequence:

```
relaxation
    set echo off message off end
    for $id in id (name ca) loop rfac
        print
            selection=(byres(id $id))
            threshold 9999
        end
    end loop rfac
    set echo on message on end
end
```

Different types of local R values can be calculated using the atom selection routine. For example, the R value for the region around each residue can be calculated with the around option of the atom selection command (see Section 2.15).

21.12 Grid Search for Optimal Correlation Time

The example file below shows how to set up a grid search for the correlation time that produces the lowest R value. In a similar manner, multidimensional grid searches can be set up for the correlation time and the order parameter.

```
remarks file: nmr_relaxation/taugrid.inp

structure @c2.psf end                                  {*Read structure file.*}
coor @c2_01_sa.pdb                                       {*Read coordinates.*}

relaxation
    nrestraints 200                          {*Expected number of restraints.*}

{*========================== Specify the physical parameters of the system.*}

    group      DMSO                 {*A group of spectra, DMSO, is defined.*}
    unresolved DMSO
        methyl                               {*Methyl protons are unresolved.*}
    end
    average 3                                 {*R-3 average to methyl groups.*}
    omega      DMSO 500.0e6                  {*Spectrometer frequency, in Hz.*}

    classification D300                      {*Define a classification, D300,*}
    @c2_300.int

    clgroup     D300 DMSO                    {*which belongs to group DMSO.  *}
    taumix      D300 0.30                                {*Mixing time, in s.*}
    minint      D300 0.05                   {*Minimum measurable intensity.*}

    cutoff               {*Cutoff used for gradient and relaxation matrix.*}
        mode all  value 4.5
    end

    potential parabola                       {*Error estimates are not used.*}
    iexp 0.1666666667                      {*Dev = I_calc^iexp - I_obs^iexp.*}
end

set display c2_taugrid.disp end

eval ($tau = 0.5e-9)
while ($tau < 5.0e-9) loop tau

    relax

       taucorrel  DMSO                             {*Correlation times, in s.*}
           model lipari                                   {*Order parameters.*}
           vector (name hn or name ha)
                (name hn or name ha)                      $tau 0.85
           vector (name hn or name ha)
                (name h* and not (name hn or name ha))    $tau 0.80
           vector (name h* and not (name hn or name ha))
                (name h* and not (name hn or name ha))    $tau 0.65
       end

       calibrate             {*Calculate spectrum and scale to experiment.*}
           quality 0.33  automatic on  reference all  group class
       end

       print threshold= 9999 end
    end

    display $tau $RNMR

    eval ($tau = $tau + 0.25e-9)
end loop tau
```

```
61
62
63 stop
```

22 Installation

This chapter describes how to install X-PLOR and how to change the array dimensions of the program. X-PLOR has been tested extensively on VAX/VMS, CRAY-YMP, CONVEX C220, STELLAR GS2000, SGI 4D/220, and NeXT. These machines are referred to as "supported." A large number of "unsupported" installation procedures are supplied on the X-PLOR distribution tape as well, in particular for FPS 511EA Series, IBM 3090 (CMS), ALLIANT, CYBER205, IBM6000, ESV, SUN, and HP720. These installation procedures are presented here courtesy of various X-PLOR users. They have not been tested with the current version and may require modification. The supported installation procedures should provide sufficient information if the user runs into problems.

22.1 Precompilation

X-PLOR is written in standard FORTRAN-77, with the exception of the following statements:

```
DO <do-loop-counter>
  <FORTRAN-statements>
END DO

DO UNTIL (<logical expression>)
  <FORTRAN-statements>
END DO

DO WHILE (<logical expression>)
  <FORTRAN-statements>
END DO

INCLUDE 'filename'

IMPLICIT NONE
```

The meaning of these statements is self-evident. They are converted to standard FORTRAN-77 by the PREXPLOR precompiler. In addition, the

inclusion of common block files with the INCLUDE statement is carried out by the PREXPLOR precompiler. The PREXPLOR precompiler also modifies the array dimensions of X-PLOR if this is desired. PREXPLOR reads the dimensioning information from a single file ("prexplor.dim") that contains the dimensioning parameters one wants to modify. In addition, PREXPLOR changes the variables of X-PLOR from single to double precision or from double to single, depending on the machine. There are special versions of PREXPLOR available for the machines mentioned below.

A printing option to make a printout of the source code utilizes the command file "pretty.com".

22.2 Dimensioning

The precompiler PREXPLOR makes use of a template file (called "PREXPLOR.DIM") to adjust the dimensioning of the X-PLOR program. If any dimension is exceeded during execution of X-PLOR, the user has to modify "PREXPLOR.DIM" in the appropriate machine-dependent directory. The fact that an array dimension is exceeded is indicated by an error message of the form

```
%<subroutine>-ERR: ABCD is exceeded.
```

The program terminates execution in batch mode and it will produce a severe error message in interactive mode. The user has to modify the specified parameter in the "prexplor.dim" file and reinstall the program as described in the previous section.

Error messages can occur when the quasi-dynamic HEAP space is exceeded. On machines with static memory management, the "HEAPDM" parameter has to be increased. On most machines, the heap is automatically expanded until the site-dependent maximum memory allocation is exceeded. In that case, the site-dependent quota (such as PAGEFILE quota on VAX) has to be increased, or the use of the HEAP has to be reduced. One way to reduce the needed HEAP space during any kind of structure factor calculation is to reduce the "MEMOry" specification within the xrefin FFT statement (Section 12.3).

22.3 Machine-dependent Routines

The use of machine-dependent features has been minimized in X-PLOR. There are, however, a few features that cannot be made portable: time, date, username, system-name, file open, file close, file inquiry, floating-point traps, interactive prompting, and optimized 3-dimensional FFT. With the exception of the FFT routines, all these features are straightforward to obtain on any machine. The vectorization of X-PLOR was carried out in such a

way that it does not degrade performance on conventional machines to a significant degree. Machine-dependent routines reside in modules "vaxtime.s", "craytime.s", "convextime.s", and so on.

Please make sure that you have copied the machine-dependent ".s" file from the machine-specific directory into the source directory before invoking the installation procedure.

22.4 Fast Fourier Transformation Routines

X-PLOR uses a fast Fourier transformation method for the computation of the structure factors and their first derivatives. High efficiency is achieved on supercomputers by the use of special assembly-coded routines.

In the case of CRAY computers, the 3-dimensional FFT calculation is reduced to a series of simultaneous (multiple) 1-dimensional FFT calculations, accomplishing a high degree of vectorization. The complex-to-complex multiple FFT routine CFFTMLT was used, which is available in SCILIB. (SCILIB is a product of CRAY Research Inc.) The routine CFFTMLT requires an initialization, which is provided by the routine CFTFAX. The CFFTMLT routine also requires that the three dimensions of the FFT be multiples of 2, 3, and 5. To avoid memory contingency problems, the physical dimension of the 3-D matrix is increased by one. These requirements are automatically satisified in X-PLOR by the routine FFTPRP, which resides in CRAYTIME.S. However, the user can manually change the parameters by entering the "XREFin FFT" command level and assigning the appropriate new parameters.

In the case of CONVEX computers, the 3-dimensional FFT calculation is accomplished by using a routine called Z3DFFT that resides in VECLIB. (VECLIB is a product of CONVEX Computer Corp.) VECLIB requires no special provisions during link. The user should check the module CONVEXTIME.S if any problems arise. The Z3DFFT routine requires that the three dimensions of the FFT be multiples of 2, 3, and 5. To avoid memory contingency problems, the physical dimension of the 3-D matrix is increased by one. These requirements are automatically satisified in X-PLOR by the routine FFTPRP, which resides in CONVEXTIME.S. However, the user can manually change the parameters by entering the "XREFin FFT" command level and assigning the appropriate new parameters.

In the case of SGI computers, special routines are supplied in the directory "machine/supported/sgi/fft". These routines need to be compiled and linked to X-PLOR. To do so, follow the installation script file.

In the case of all other computers, source code for the Temperton FFT routines is supplied in the corresponding machine-specific module.

22.5 Test Suite

Over 80 input and output files are provided in the "test" directory. They can be used to check the correctness of a particular installation, a useful procedure when one is installing X-PLOR on new systems with optimizing FORTRAN compilers. The "runtests" and "difftests" procedures should be used before running the test cases; they copy the necessary parameter and topology files from the "toppar" directory into the "test" directory.

22.6 Directory Structure on Distribution Medium

X-PLOR is distributed on a VAX/VMS (backup) or UNIX (tar) tape. The following is a list of all subdirectories:

[xplor]

[xplor.benchmark1]

[xplor.benchmark2]

[xplor.machine]

[xplor.machine.supported]

[xplor.machine.supported.convex]

[xplor.machine.supported.cray-unicos]

[xplor.machine.supported.next]

[xplor.machine.supported.sgi]

[xplor.machine.supported.stellar]

[xplor.machine.supported.vax]

[xplor.machine.unsupported]

[xplor.machine.unsupported.aix370]

[xplor.machine.unsupported.alliant_FX2800]

[xplor.machine.unsupported.alliant_FX40]

[xplor.machine.unsupported.cray-cos]

[xplor.machine.unsupported.cyber205]

[xplor.machine.unsupported.esv]

[xplor.machine.unsupported.fps]

[xplor.machine.unsupported.hp720]

[xplor.machine.unsupported.ibm]

[xplor.machine.unsupported.ibmrios]

[xplor.machine.unsupported.ibm6000]

[xplor.machine.unsupported.sun]

[xplor.machine.unsupported.titan]

[xplor.source]

[xplor.stellar]

[xplor.symmetry]

[xplor.test]

[xplor.toppar]

[xplor.symmetry]

[xplor.tutorial]

22.7 VAX/VMS Installation

Load the "vax", "source", "toppar", "symmetry", and "tutorial" directories. Go into VAX directory and modify "login.com" according to your directory structure. Execute "login.com". Compile and link "prexplor.for" by executing "prexplor_make.com". Execute "install.com" (this will create the X-PLOR object library), and then compile the source and link the executable. To modify dimensions, edit file "prexplor.dim" and recompile and link everything (use "compile.com" and "link.com"). Machine-dependent routines of X-PLOR are in "vaxtime.s". The precompiler is in "prexplor.for". To execute the program, type XPLOR. Make sure that you have copied "vaxtime.s" into the source directory before running the installation procedure. The directory structure and usage are described above.

Minimal Directory Structure to Test and Execute X-PLOR

Other directories than those in the following list (such as the "tutorial" directory) should be created upon demand.

[xplor.]

[xplor.benchmark1]

[xplor.benchmark2]

[xplor.source]

[xplor.symmetry]

[xplor.test]

[xplor.toppar]

[xplor.vax]

Command Files (Located in Directory [xplor.vax])

compile.com executes the put.com command file with each ".s" file in the $SOURCE directory.

difftests.com notes differences of all ".inp" files in the $TEST directory.

install.com executes the command files in the sequence required to perform a complete X-PLOR installation.

link.com links X-PLOR using the "xplor.olb" library (in the "vax" directory).

login.com defines all symbols and logicals.

pretty.com produces a listing of the specified ".s" source file.

prexplor_make.com makes the "prexplor" executable file.

Put.com filename [debug flag] precompiles, compiles, and inserts in the library the given filename. It uses library "xplor.olb" in directory [xplor.vax] and deletes intermediate files.

runtests.com runs X-PLOR once for each test case in the $TEST directory. The output from these tests resides in the "*.vax" files.

Symbol Definitions

Logicals

$COMS XPLOR:[vax] directory (contains command files)

$OBJ XPLOR:[vax] location of object library

$SOURCE XPLOR:[source] directory (contains source files)

$SYMMETRY XPLOR:[symmetry] directory (contains crystallographic files)

$TEST XPLOR:[test] directory (contains test files)

$TOPPAR XPLOR:[toppar] directory (contains topology files)

$XPLOR concealed device name

Symbols

compile @compile.com

difftests @difftests.com

link @link.com

pretty @pretty.com

put @put.com

runtests @runtests.com

xplor run/nodebug cd:xplor.exe

22.8 Generic UNIX Installation

This section is applicable not only to the CONVEX but to all UNIX installations of X-PLOR.

Load the "convex", "source", "toppar", "symmetry", and "tutorial" directories on the system. Make the "object_library" directory. Machine-dependent routines are in "convextime.s" and "convex.c". **Make sure you have copied the "convextime.s" file into the SOURCE directory**. The FORTRAN code for the precompiler "PREXPLOR" is in the file "prexplor.f". Compile and link "prexplor.f" by executing "prexplor_make.com". Dimensioning is stored in "prexplor.dim". All command files in the "convex" directory are for the C-shell. Go into the "convex" directory and modify "ulogin.com" according to your directory structure. Execute (source) ulogin.com in the C-shell. Make sure the "convextime.s" module is in the $SOURCE directory. Issue the install command (to run it in background, do "install >& job.log &"). The install command will compile all "*.s" files as well as the machine-dependent c module (convex.c).

Minimal Directory Structure to Test and Execute X-PLOR

Other directories than those in the following list (such as the "tutorial" directory) should be created upon demand.

xplor/

xplor/benchmark1

xplor/benchmark2

xplor/convex

xplor/object_library

xplor/symmetry

xplor/source

xplor/test

xplor/toppar

Command Files (Located in Directory xplor/convex)

compile.com [filemask1 [filemask2 [...filemaskn]]] compiles all files matching supplied file masks using maximum optimization level.

difftests.com [filemask1 [filemask2 [...filemaskn]]] compares the test output files in xplor/test directory, matching the supplied file masks with the same name and extensions of "*.convex" and "old/*.convex". It then removes all lines that are expected to differ because of times or filename paths. The result is left in the file of the given name with the extension ".diff".

install.com executes the command files in the sequence required to perform a complete X-PLOR installation.

link.com links X-PLOR using the FORTRAN compiler (fc). All object files must reside in the $OBJ directory.

precompile.com [filemask1 [filemask2 [...filemaskn]]] precompiles all files in the current working directory and any subdirectories that match the given file mask(s). The output from the precompilation resides in the file with the same name, except for the extension, which is changed to "p".

pretty.com [filemask1 [filemask2 [...filemaskn]]] produces a listing of the specified ".s" source file.

prexplor_make.com makes the "prexplor" executable file.

put.com [filemask1 [filemask2 [...filemaskn]]] performs both the **Precompile** and **Compile** commands.

runtests.com [filemask1 [filemask2 [...filemaskn]]] runs X-PLOR once for each test case in the xplor/tests directory. The output from these resides in the ".convex" files.

ulogin.com defines all environment variables and aliases.

Environment Variables & Aliases

Unix Variables

$COMS xplor/convex directory (contains command files)

$OBJ xplor/object_library directory (contains object files)

$SOURCE xplor/source directory (contains source files)

$SYMMETRY xplor/symmetry directory (contains crystallographic files)

$TEST xplor/test directory (contains test files)

$TOPPAR xplor/toppar directory (contains topology files)

$XPLOR X-PLOR directory

Aliases

compile $COMS/compile.com

difftests $COMS/difftests.com

install $COMS/install.com

link $COMS/link.com

precompile $COMS/precompile.com

pretty $COMS/pretty.com

put $COMS/put.com

runtests $COMS/runtests.com

xplor $OBJ/xplor.exe (executes X-PLOR program)

Notes and Restrictions

The commands **put, precompile,** and **compile** require that the current working directory contain any and all sources to be compiled. They will accept a file mask or masks as argument(s). These commands do not perform extensive error checking, and unpredictable results can be expected if they are used out of context.

22.9 CRAY Installation

Load the "cray", "source", "toppar", "symmetry", and "tutorial" directories on the UNICOS system. Go into the "cray" directory and modify "ulogin.com" according to your directory structure. Installation and execution are similar to the CONVEX C1/C2 installation of X-PLOR (see Section 22.8). Use of the special FFT routines is similar to the CRAY-COS installation of X-PLOR (routine CFFTMLT in SCILIB is used). HEAP allocation is dynamic. The CFT77 compiler is recommended for all modules.

22.10 SGI Installation

The installation on the SGI proceeds analogously to the CONVEX and UNICOS systems. (Follow the instructions in Section 22.8, but use the "sgi" directory.)

22.11 NeXT Installation

The installation on the NeXT proceeds analogously to the CONVEX and UNICOS systems. (Follow the instructions in Section 22.8, but use the "next" directory.) This installation requires the ABSOFT FORTRAN compiler.

22.12 Benchmark Results

The following table lists the CPU times in seconds for four benchmark jobs with X-PLOR that were run on a number of machines.

Task	*CRAY-YMP*	*Convex-C210* 2 mcm	*STELLAR* GS2000	*Alliant* 4CEs	*CRAY C-90* 3000
1	33.5	225.2	531.	442.2	22.24
2	1.25	8.3	28.	31.4	0.758
3	0.85	8.1	15.	33.1	0.284
4	0.38	1.9	2.	6.2	0.111

Task	*SGI R4400*	*SGI R4000* 3100 (risc)	*VAX8700*	*FPS 511EA*
1	148.4	233.	2504.5	268.7
2	4.8	7.6	86.8	12.2
3	8.9	14.3	348.3	6.2
4	2.5	3.6	32.8	2.5

Task	*IBM 6000/580*	*HP 9000/735*	*NeXT (68040)*	IBM 3090(CMS)	DEC alpha (model
1	113.4	71.	1828.1	211.4	89.9
2	6.	2.6	44.	8.4	2.89
3	16.8	7.07	90.0	34.2	2.92
4	1.5	2.65	18.0	3.6	1.5

Benchmark 1 (see Chapter 22) consists of task 1. Benchmark 2 consists of tasks 2–4. Task 1 is a 0.5-psec MD-simulation on a DNA-undodecamer, task 2 is an electron density calculation at 2.8 Å for 3086 atoms, task 3 is a 3-d real fast Fourier transformation on a 180 x 96 x 96 grid, and task 4 is the application of four symmetry operators in reciprocal space to 8783 reflections. (Mcm stands for memory controller pair on the CONVEX.) All benchmarks were run in 64-bit precision. Task 1 has short vector loops; tasks 3 and 4 have long vector loops. Task 2 is predominantly scalar. No machine-specific modifications of the X-PLOR main source were carried out. DEC-alpha uses the DXML FFT library routines. HP 735 uses the MLIB FFT library routines.

Index of Syntactic Definitions

<**minimize-rigid-statement**> Section 9.2

<**mmdg-statement**> Section 19.1.1

<**nbonds-statement**> Section 3.2.1

<**NCS-restraints-group-statement**> Section 16.1

<**NCS-restraints-statement**> Section 16.1.1

<**NCS-strict-statement**> Section 16.2.1

<**noe-statement**> Section 18.1

<**one-character-word**> Section 2.1

<**op**> Section 2.14

<**operation**> Section 2.14

<**orient-statement**> Section 11.4

<**parameter-statement**> Section 3.2.1

<**patch-character**> Sections 3.1.1 and 3.8

<**patch-statement**> Section 3.8

<**pdb-record**> Section 6.1

<**pick-statement**> Section 5.1.2

<**PREDict_intensities-statement**> Section 21.1.1

<**print-deviation-statement**> Section 21.1.1

<**print-objects**> Section 5.1.1

<**print-statement**> Section 5.1.1

<**property**> Section 2.15

<**read-trajectory-statement**> Section 11.2

<**real**> Section 2.2

<**reflection-property**> Section 12.4

<**relaxation-statement**> Section 21.1.1

<**residue-name**> Section 3.1.1

<**residue-number**> Section 3.7

<**residue-statement**> Section 3.1.1

<**restraints-dihedral-statement**> Section 7.2

<**restraints-harmonic-statement**> Section 7.1

<**restraints-planar-statement**> Section 7.3

<**rotman-statement**> Section 17.2

<**segment-name**> Section 3.7

<**segment-statement**> Section 3.7

<**selection**> Section 2.15

<**selection-expression**> Section 2.15

<**set-statement**> Section 2.13

<**shake-statement**> Section 8.2

<**shell-statement**> Section 11.9

<**solmask-statement**> Section 12.7

<**string**> Section 2.2

<**structure-statement**> Section 3.11

<**surface-statement**> Section 5.6

<**symbol**> Section 2.5

<**TAUCorrelation-statement**> Section 21.1.1

<**term**> Section 2.15

<**topology-statement**> Section 3.1.1

<**trajectory-statement**> Section 11.1.3

<**type**> Section 3.1.1

<**unresolved-statement**> Section 21.1.1

<**vector-expression**> Section 2.16

<**vector-mode**> Section 2.16

<**vector-operation**> Section 2.16

<**vector-show-property**> Section 2.16

<**vector-statement**> Section 2.16

<**vflc**> Section 2.14

<**wildcard**> Section 2.6

<**word**> Section 2.1

<**write-coordinates-statement**> Section 6.2

<**write-parameter-statement**> Section 3.3

<**write-reflection-statement**> Section 12.4

<**write-structure-statement**> Section 3.12

<**write-trajectory-statement**> Section 11.3

<**X-PLOR-statement**> Section 2.8

<**xrefin-do-expression**> Section 12.5

<**xrefin-do-mode**> Section 12.5

<**xrefin-do-statement**> Section 12.5

<**xrefin-function**> Section 12.5

<**xrefin-map-statement**> Section 14.1

<**xrefin-optimize-bfactor-statement**> Section 13.4

<**xrefin-optimize-group-statement**> Section 13.3

<**xrefin-optimize-overall-statement**> Section 13.2

<**xrefin-property**> Section 12.5

<**xrefin-reflection-statement**> Section 12.4

<**xrefin-search-rotation-statement**> Section 17.1

<**xrefin-search-translation-statement**> Section 17.3

<**xrefin-statement**> Section 12.3

<**xrefin-vflc**> Section 12.5

List of Application Statements

The following is a list of the application statements that are accessible from the main level of X-PLOR:

<application-statement>:==

CONStraints FIX <constraints-fix-statement> END

CONStraints { INTEr <constraints-interaction-statement> } END

COORdinate <coordinate-statement> END

DELEte { <delete-statement> } END

DISTance { <distance-statement> } END

DUPLicate { <duplicate-statement> } END

DYNAmics ANALyze <dynamics-analyze-statement> END

DYNAmics MERGe { <dynamics-merge-statement> } END

DYNAmics RIGId { <dynamics-rigid-statement> } END

DYNAmics VERLet { <dynamics-Verlet-statement> } END

ENERgy { <energy-statement> } END

FLAGs { <flag-statement> } END

HBUIld { <hbuild-statement> } END

MINImize POWEll { <minimize-powell-statement> } END

MINImize RIGId { <minimize-rigid-statement> } END

MMDG { <mmdg-statement> } END

NOE { <noe-statement> } END

PARAmeter { <parameter-statement> } **END**

PATCh <patch-statement> **END**

PICK <pick-statement>

PRINt <print-statement>

READ TRAJectory { <read-trajectory-statement> } **END**

RELAxation { <relaxation-statement> } **END**

RESTraints DIHE { <restraints-dihedral-statement> } **END**

RESTraints HARM { <restraints-harmonic-statement> } **END**

RESTraints PLANar { <restraints-planar-statement> } **END**

SEGMent { <segment-statement> } **END**

SHAKe { <shake-statement> } **END**

STRUcture { <structure-statement> } **END**

SURFace { <surface-statement> } **END**

TOPOlogy { <topology-statement> } **END**

VECTor <vector-statement>

WRITe COORdinates { <write-coordinates-statement> } **END**

WRITe STRUcture { <write-structure-statement> } **END**

WRITe TRAJectory { <write-trajectory-statement> } **END**

XREFin { <xrefin-statement> } **END**

Bibliography

Berendsen, H.J.C., Postma, J.P.M., van Gunsteren, N.F., DiNola, A., and Haak, J.R. (1984). "Molecular dynamics with coupling to an external bath", *J. Chem. Phys.* 81, 3684–3690.

Bernstein, R., Ross, A., Cieslar, C., and Holak, T. A. (1992). "A Practical Approach to Calculations of Biomolecular Structures from Homonuclear Three-Dimensional NOE-NOE Spectra", *J. Magn. Reson.*, in press.

Bonvin, A.M.J.J. and Brünger, A.T. (1994). "Conformational Variability in Solution Nuclear Magnetic Structures", *J. Mol. Biol.* 250, 80-93.

Bricogne, G. (1976). "Methods and Programs for Direct-Space Exploitation of Geometric Redundancies", *Acta Cryst.* A32, 832–847.

Brooks, B., Bruccoleri, R., Olafson, B., States, D., Swaminathan, S., and Karplus, M. (1983). "CHARMM: A Program for Macromolecular Energy, Minimization, and Molecular Dynamics Calculations", *J. Comp. Chem.* 4, 187–217.

Brünger, A.T. (1988). "Crystallographic Refinement by Simulated Annealing: Application to a 2.8 Å Resolution Structure of Aspartate Aminotransferase", *J. Mol. Biol.* 203, 803–816.

Brünger, A.T. (1989). "A Memory-Efficient Fast Fourier Transformation Algorithm for Crystallographic Refinement on Supercomputers", *Acta Cryst.* A45, 42–50.

Brünger, A.T. (1990). "Extension of Molecular Replacement: A New Search Strategy based on Patterson Correlation Refinement", *Acta Cryst.* A46, 46–57.

Brünger, A.T., Milburn, M.V., Tong, L., de Vos, A.M., Jancarik, J., Yamaizumi, Z., Nishimura, S., Ohtsuka, E. & Kim, S.-H. (1990b). "Crystal Structure of an Active Form of *ras* Protein, a Complex of GTP Analog and c-H-*ras* P21 Catalytic Domain", *Proc. Natl. Acad. Sci. USA* **87**, 4849–4853.

Brünger, A.T. (1991a). "Crystallographic Phasing and Refinement of Macromolecules", *Current Opinion in Structural Biology* 1, 1016–1022.

Brünger, A.T. (1991b). "Simulated Annealing in Crystallography", *Ann. Rev. Phys. Chem.* 42, 197–223.

Brünger, A.T. (1991c). "Solution of a Fab (26-10) /Digoxin Complex by Generalized Molecular Replacement", *Acta Cryst.* A47, 195–204.

Brünger, A.T. (1992). "The Free R value: a Novel Statistical Quantity for Assessing the Accuracy of Crystal Structures", *Nature* 355, 472–474.

Brünger, A.T. (1993). "Assessment of Phase Accuracy by Cross Validation: The Free R Value. Methods and Applications", *Acta Cryst. D*, in press.

Brünger, A.T., Brooks, C.L., and Karplus, M. (1984). "Stochastic Boundary-Conditions for Molecular-Dynamics Simulations of ST2 Water", *Chem. Phys. Lett.* 105, 495–500.

Brünger, A.T., Clore, G.M., Gronenborn, A.M., and Karplus, M. (1986). "Three-dimensional structures of proteins determined by molecular dynamics with interproton distance restraints: Application to crambin", *Proc. Natl. Acad. Sci. USA* 83, 3801–3805.

Brünger, A.T., Clore, G.M., Gronenborn, A.M., and Karplus, M. (1987). "Solution conformations of a human growth hormone releasing factor: comparison of the restrained molecular dynamics and distance geometry methods for a system without long-range distance data", *Protein Engineering* 1, 399–406.

Brünger, A.T. and Karplus, M. (1991). "Molecular Dynamics Simulations with Experimental Restraints", *Accounts Chemical Research* 24 , 54–61.

Brünger, A.T. and Karplus, M. (1988). "Polar hydrogen positions in proteins: Empirical energy function placement and neutron diffraction comparison", *Proteins* 4,148–156.

Brünger, A.T., Karplus, M., and Petsko, G.A. (1989). "Crystallographic Refinement by Simulated Annealing: Application to a 1.5 Å Resolution Structure of Crambin", *Acta Cryst.* A45, 50–61.

Brünger, A. T., Krukowski, A., and Erickson, J. (1990). "Slow-Cooling Protocols for Crystallographic Refinement by Simulated Annealing", *Acta Cryst.* A46, 585–593.

Brünger, A.T., Kuriyan, J., and Karplus, M. (1987). "Crystallographic R Factor Refinement by Molecular Dynamics", *Science* 235, 458–460.

Brünger, A.T., Leahy, D.J., Hynes, T.R., and Fox, R.O. (1991). "The 2.9 Å Resolution Structure of an Anti-Dinitrophenyl-Spin-Label Monoclonal Antibody Fab Fragment with Bound Hapten", *J. Mol. Biol.* 221 , 239–256.

Brukert, U. and Allinger, N.L. (1982). In: *Molecular Mechanics*, ACS Monograph 177, American Chemical Society, Washington.

Clore, G.M., Appella, E., Yamada, M., Matsushima, K., and Gronenborn, A.M. (1990). "Three-Dimensional Structure of Interleukin 8 in Solution", *Biochemistry* 29, 1689–1696.

Clore, G.M. and Gronenborn, A.M. (1989). "Determination of three-dimensional structures of proteins and nucleic acids in solution by nuclear magnetic resonance spectroscopy", *CRC Critical Reviews in Biochemistry* 24, 479–564.

Clore, G.M. and Gronenborn, A.M. (1991). "Structures of Larger Proteins in Solution: Three- and Four-Dimensional Heteronuclear NMR Spectroscopy", *Science* 252, 1390–1399.

Cremer, D. and Pople, J.A. (1975). "General Definition of Ring Puckering Coordinates", *J. Am. Chem. Soc.* 97, 1354–1358.

Crippen, G. and Havel T. (1988). *Distance Geometry and Molecular Conformation.* Research Studies Press: Taunton, Somerset, England.

DeLano, W.L. and Brünger, A.T. (1994). *Helix packing in proteins: prediction and energetic analysis of dimeric, trimeric and tetrameric GCN4 coiled coil structures. PROTEINS: Struct., Funct. & Genetics* 20, 105-123.

Engh, R.A. and Huber, R. (1991). "Accurate Bond and Angle Parameters for X-ray Protein-Structure Refinement", *Acta Cryst.* A47, 392–400.

Ernst, R.R., Bodenhausen, G., and Wokaun, A. (1987). *Principles of Nuclear Magnetic Resonance in One and Two Dimensions*. Oxford: Clarendon Press.

Goldstein, H. (1980). *Classical Mechanics.* New York: Addison-Wesley, chap. 4.

Gronenborn, A.M., Filpula, D.R., Essig, N.Z., Achari, A., Whitlow, M., Wingfield, P.T., and Clore, G.M. (1991). "The immunoglobulin binding domain of streptococcal protein G has a novel and highly stable polypeptide fold", *Science* 253, 657–661.

Ha, S.N., Giammona, A., Field, M. and Brady, J.W. (1988). "A Revised Potential Energy Surface for Molecular Mechanics Studies of Carbohydrates", *Carbohydrate Res.* 180, 207–221.

Habazettl, J., Cieslar, C., Oschkinat, H., and Holak, T.A. (1990). "H-1-NMR Assignments of Side-Chain Conformations in Proteins Using a High-Dimensional Potential in the Simulated Annealing Calculations", *FEBS Lett.* 268, 141–145.

Habazettl, J., Ross, A., Oschkinat, H. and Holak, T. A. (1992a). "Secondary NOE pathways in 2D NOESY spectra of proteins estimated from homonuclear three-dimensional NOE-NOE nuclear magnetic resonance spectroscopy", *J. Magn. Reson.* 97, 281–294.

Habazettl, J., Schleicher, M., Otlewski, J., and Holak, T. A. (1992b). "Homonuclear Three-Dimensional NOE-NOE NMR Spectra for Structure Determination of Proteins in Solution", *J. Mol. Biol.*, in press.

Hahn, T., ed. (1987). *International Tables for Crystallography*, Vol. A. , D. Reidel Publishing Company.

Havel, T.F. and Wüthrich, K. (1984). "A distance geometry program for determining the structures of small proteins and other macromolecules from nuclear magnetic resonance measurements of intramolecular ^{1}H^{1}H proximities in solution" *Bull. Math. Bio.* 46, 673–698.

Head-Gordon, T. and Brooks, C.L. (1991). "Virtual Rigid Body Dynamics", *Biopolymers* 31, 77–100.

Hendrickson, W.A. (1985). "Stereochemically Restrained Refinement of Macromolecular Structures", *Meth. Enzymol.* 115, 252–270.

Hendrickson, W.A. and Ward, K.B. (1976). "A Packing Function for Delimiting the Allowable Locations of Crystallized Macromolecules", *Acta Cryst.* A32, 778–780.

Hirshfeld, F.L. (1968). "Symmetry in the Generation of Trial Structures", *Acta Cryst.* A 24, 301–311.

Hodel, A., Kim, S.-H., and Brünger, A.T. (1992). "Model Bias in Macromolecular Crystal Structures", *Acta Cryst. A*, in press.

Huber, R. (1985). "Experience with the application of Patterson search techniques", *Molecular Replacement*, (Machin, P.A., comp.), Proceedings of the Daresbury Study Weekend, Science and Engineering Research Council, The Librarian, Daresbury Laboratory, Daresbury, United Kingdom, 58–61.

James, T.L., Gochin, M., Kerwood, D.J., Schmitz, U., and Thomas, P.D. (1991). In: *Computational Aspects of the Study of Biological Macromolecules by NMR*, (J.C. Hoch, ed.). New York: Plenum Press.

Jorgensen, W., Chandrasekar, J., Madura, J., Impey, R., and Klein, M. (1983). "Comparison of simple potential functions for simulating liquid water", *J. Chem. Phys.* 79, 926–935.

Jorgensen, W. and Tirado-Rives, J. (1988). "The OPLS potential function for proteins, energy minimization for crystals of cyclic peptides and crambin", *J. Am. Chem. Soc.* 110, 1657–1666.

Kabsch, W. (1976). "A Solution for the Best Rotation to Relate Two Sets of Vectors.", *Acta Cryst.* A32, 922.

Kaptein, R., Zuiderweg, E.R.P., Scheek, R.M., Boelens, R., and van Gunsteren, W.F. (1985). "A protein structure from NMR data. lac Repressor headpiece", *J. Mol. Biol.* 182, 179–182.

Karplus, M. and Petsko, G.A. (1990). "Molecular-Dynamics Simulations in Biology", *Nature* 347, 631–39.

Keepers, J. W. and James, T. L. (1984). "A Theoretical Study of Distance Determinations from NMR. Two-Dimensional Nuclear Overhauser Effect Spectra", *J. Magn. Reson.* 57, 404–426.

Kirkpatrick, S., Gelatt, C.D.Jr., Vecchi, M.P. (1983). "Optimization by Simulated Annealing", *Science* 220, 671–680.

Koch, E. & Fischer, W. (1987). Euclidean and affine normalizers of space groups and their use in crystallography. *International Tables for Crystallography, Volume A, Space Group Symmetry.* (T. Hahn, T, ed.), Dordrect: D. Reidel Publishing company, 853–869.

Wolfram, S. (1991). *Mathematica, A System for Doing Mathematics by Computer*, 2nd Ed. Redwood City, California: Addison-Wesley Publishing Company, Inc.

Kuszewski, J., Nilges, M., and Brünger A.T. (1992). "Sampling and efficiency of metric matrix distance geometry: A novel "partial" metrization algorithm", *J. Biomol. NMR* 2, 33-56.

Lattman, E.E. (1985). "Use of the Rotation and Translation Functions", *Methods Enzymol.* 115, 55–77.

Lee, B. and Richards, F.M. (1971). "The Interpretation of Protein Structures: Estimation of Static Accessibility", *J. Mol. Biol.* 55, 379–400.

Lifson, S. and Stern, P.S. (1982). "Born-Oppenheimer Energy Surfaces of Similar Molecules – Interrelations Between Bond Lengths, Bond Angles, and Frequencies of Normal Vibrations in Alkanes", *J. Chem. Phys.* 77, 4542–4550.

Lipari, G. and Szabo, A. (1982). "Model-Free Approach to the Interpretation of Nuclear Magnetic-Resonance Relaxation in Macromolecules .1. Theory and Range of Validity", *J. Am. Chem. Soc.* 104, 4546–4559.

Luzzati, P.V. (1952). "Traitement Statistique des Erreurs dans la Determination des Structures Cristallines", *Acta Cryst* 5, 802–810.

Macura, S. and Ernst, R. R. (1980). "Elucidation of Cross Relaxation in Liquids by Two-Dimensional N.M.R. Spectroscopy", *Mol. Phys.* 41, 95–117.

Main P. (1975). "Recent Developments in the MULTAN System – The Use of Molecular Structure", In "Crystallographic Computing Techniques", (Ahmed, F.R., Huml, K., Sedláček, B., eds.), 98-103.

Morris, L., MacArthur, M.W., Hutchinson, E.G., and Thornton, J.M. (1992). Stereochemical Quality of Protein Structure Coordinates, *Proteins* 12, 345–364.

Naur, P. (1960). "Revised Report on Algorithmic Language Algol-60", *Computer J.*, 5, 349.

Némethy, G., Pottie, M.S. and Scheraga, H.A. (1983) "Energy Parameters in Polypeptides. 9. Updating of Geometrical Parameters, Nonbonded Interactions, and Hydrogen Bond Interactions for the Naturally Occurring Amino Acids", *J. Phys. Chem.* 87, 1883–1887.

Nilges, M. and Brünger, A.T. (1991), *J. Cellular Biochem.*, Supplement 15G, Abstract, CG 422, 98.

Nilges, M., Clore, G.M., and Gronenborn, A.M. (1988a). "Determination of three-dimensional structures of proteins from interproton distance data by dynamical simulated annealing from a random array of atoms", *FEBS Lett.* 239, 129–136.

Nilges, M., Clore, G.M., and Gronenborn, A.M. (1988b). "Determination of three-dimensional structures of proteins from interproton distance data by hybrid distance geometry-dynamical simulated annealing calculations", *FEBS Lett.* 229, 317–324.

Nilges, M., Gronenborn, A.M., Brünger, A.T., and Clore, G.M. (1988c). "Determination of Three-Dimensional Structures of Proteins by Simulated Annealing with Interproton Distance Restraints. Application to Crambin, Potato Carboxypeptidase Inhibitor and Barley Serine Proteinase Inhibitor 2", *Protein Engineering* **2**, 27–38.

Nilges, M., Gronenborn, A.M., and Clore, G.M. (1989). "Structure determination from NMR conformational data by molecular dynamics calculations", *Proceedings of the 1989 Daresbury Study Weekend.* Science and engineering research council, Daresbury Laboratory, Warrington, WA4AA0, UK.

Nilges, M., Habazettl, J., Brünger, A.T., and Holak, T.A. (1991). "Relaxation Matrix Refinement of the Solution Structure of Squash Trypsin-Inhibitor", *J. Mol. Biol.* 219, 499–510.

Nilges, M., Kuszewski, J., and Brünger, A.T. (1991). In: *Computational Aspects of the Study of Biological Macromolecules by NMR*, (J.C. Hoch, ed.). New York: Plenum Press.

Nilsson, L., Clore, G.M., Gronenborn, A.M., Brünger, A.T., and Karplus, M. (1986). "Structure refinement of oligonucleotides by molecular dynamics with nuclear Overhauser effect interproton distance restraints: Application to 5' d(C-G-T-A-C-G)$_2$", *J. Mol. Biol.* 188, 455–475.

Nilsson, L. and Karplus, M. (1986). "Empirical Energy Functions for Energy Minimization and Dynamics of Nucleic-Acids", *J. Comp. Chem.* 7, 591–616.

Nordman, C.E. (1980). "Procedures for Detection and Idealization of Non-crystallographic Symmetry with Application to Phase Refinement of the Satellite Tobacco Necrocis Virus Structure", *Acta Cryst.* A36, 747–754.

Pearlman, D.A. and Kollman, P.A. (1991). "Are Time-Averaged Restraints Necessary for Nuclear Magnetic Resonance Refinement? A Model Study for DNA", *J. Mol. Biol.* **220**, 457–479.

Powell, M.J.D. (1977). "Restart Procedures for the Conjugate Gradient Method", *Mathematical Programming* 12, 241–254.

Rao, S.N., Jih, J.-H., and Hartsuck, J.A. (1980). "Rotation-Function Space Groups", *Acta Cryst.* A36, 878–884.

Rogers, D. (1965). "Statistical Properties of Reciprocal Space", In "Computing Methods in Crystallography" (Rollett, J.S., ed.), 117–132.

Rossmann, M.G. and Blow, D.M. (1962). "The Detection of Sub-Units Within the Crystallographic Asymmetric Unit", *Acta Cryst.* 15, 24–31.

Ryckaert, J.-P., Ciccotti, G., and Berendsen, H.J.C. (1977). "Numerical-Integration of Cartesian Equations of Motion of a System with Constraints - Molecular-Dynamics of N-Alkanes", *J. Comput. Phys.* 23, 327–341.

Schneider, T., Nilges, M., and Brünger, A.T. (in preparation).

Schomaker, V., Waser, J., Marsh, R.E., and Bergman, G. (1959). "To Fit a Plane or a Line to a Set of Points by Least Squares", *Acta Cryst.* 12, 600–604.

Solomon, I. (1955). "Relaxation Processes in a System of 2 Spins", *Phys. Rev.* 99, 559–565.

Steinbach, P.J. and Brooks, B.R. (1994). "New Spherical-Cutoff Methods for Long-Range Forces in Macromolecular Simulation", *J. Comp. Chem.* 15, 667-683.

Stout, G.H. and Jensen, L.H. (1989). *X-ray Structure Determination*, Wiley, New York.

Strong, R., (1990). Ph.D. diss., Harvard University.

Tarjan, R. (1983). *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia.

Torda, A.E., Scheek, R.M., van Gunsteren, W.F. (1989) "Time-Dependent Distances Restraints in Molecular Dynamics Simulations", *Chem. Phys. Letters* **157**, 289–294.

Torda, A.E., Scheek, R.M., van Gunsteren, W.F. (1990) "Time-Averaged Nuclear Overhauser Effect Distance Restraints Applied to Tendamistat", *J. Mol. Biol.* **214**, 223–235.

Treutlein, H., Schulten, K., Deisenhofer, J., Michel, H., Brünger, A.T., and Karplus, M. (1992). "Chromophore-Protein Interactions and the Function of the Photosynthetic Reaction Center: A Molecular Dynamics Study", *Proc. Natl. Acad. Sci. USA* 89, 75–79.

Verlet, L. (1967). "Computer Experiments on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules", *Phys. Rev.* 159, 98–105.

Weiner, S., Kollman, P., Case, D., Singh, U.C., Ghio, C., Alagona, G., Profeta, S., and Weiner, P. (1984). "A new force field for molecular mechanical simulation of nucleic acids and proteins", *J. Am. Chem. Soc.* 106, 765–784.

Weis, W. I., Brünger, A. T., Skehel, J. J., and Wiley, D. C. (1990). "Refinement of the Influenza Virus Haemagglutinin by Simulated Annealing", *J. Mol. Biol.* 212, 737–761.

Wilson, A.J.C. (1949). "The Probability Distribution of X-ray Intensities", *Acta Cryst.* 2, 318–321.

Wolfram, S. (1991). *Mathematica, A System for Doing Mathematics by Computer*, 2nd Ed. Redwood City, California: Addison-Wesley Publishing Company, Inc.

Yip, P. and Case, D. A. (1989). "A New Method for Refinement of Macromolecular Structures based on Nuclear Overhauser Effect Spectra", *J. Magn. Reson.* 83, 643–648.

Abbreviations

amu atomic mass units

CPU central processing unit

FFT fast Fourier transformation

NCS non-crystallographic symmetry

NMR nuclear magnetic resonance

NOE nuclear Overhauser effect

PDB Brookhaven Protein Data Bank

rms root-mean-square

rmsd root-mean-square difference

Index