

MODELLER
A Program for Protein Structure Modeling
Release 6¹

Andrej Šali
with help from
András Fiser, Roberto Sánchez, Marc A. Martí-Renom,
Bozidar Jerkovic, Azat Badretdinov,
Francisco Melo, John P. Overington, and Eric Feyfant

The Rockefeller University
1230 York Avenue
New York, NY 10021, USA
tel +1-212-327 7550, fax +1-212-327 7540
email sali@rockefeller.edu
URL <http://guitar.rockefeller.edu/>

21 December, 2001

¹This version of the manual is not up to date.

Contents

Copyright notice	xiii
Acknowledgments	xv
1 Introduction	1
1.1 What is MODELLER?	1
1.2 MODELLER bibliography	2
1.3 Distribution	5
1.4 Installation	6
1.5 Bug reports	9
1.6 Method for comparative protein structure modeling by MODELLER	10
1.7 Comparative protein modeling primer	12
1.7.1 Finding structures and sequences related to the target sequence	12
1.7.2 Preparing an initial family alignment of all structures and sequences	12
1.7.3 Becoming familiar with the family fold and improving the alignment	12
1.7.4 Selecting the templates	13
1.7.5 Model building	13
1.7.6 Evaluating the models	13
1.7.7 Repeat the cycle	14
1.8 Tutorial on using MODELLER for comparative modeling	15
1.8.1 Preparing input files	15
1.8.2 Running MODELLER	16
1.8.3 Fully automated comparative modeling	16
1.8.4 Improved modeling of loops in MODELLER-6	17
1.9 Frequently asked questions (FAQ) and examples	18
1.10 MODELLER updates	29
1.10.1 Major changes in Releases 5 and 6	29
1.10.2 Detailed chronological listing of some changes since MODELLER-4 release on 17 June, 1997	29
2 MODELLER commands	33
2.1 Miscellaneous rules and features of MODELLER	33
2.1.1 MODELLER system	33
2.1.2 Running MODELLER scripts	33
2.1.3 Controlling breakpoints and the amount of output	34
2.1.4 File naming	34

2.1.5	File types	36
2.1.6	Format of the command description	36
2.2	Stereochemical parameters and molecular topology	38
2.2.1	Modeling residues with non-existing or incomplete entries in the topology and parameter libraries	38
2.2.2	READ_RESTYP_LIB — read residue type library	39
2.2.3	READ_TOPOLOGY — read residue topology library	39
2.2.4	READ_PARAMETERS — read parameters library	39
2.2.5	READ_ATOM_CLASS — read classification of atom types	40
2.2.6	GENERATE_TOPOLOGY — generate MODEL topology	40
2.2.7	PATCH — patch MODEL topology	41
2.2.8	PATCH_SS_TEMPLATES — guess MODEL disulfides from templates	42
2.2.9	PATCH_SS_MODEL — guess MODEL disulfides from model structure	44
2.2.10	MUTATE_MODEL — mutate selected MODEL residues	44
2.2.11	MAKE_TOPOLOGY_MODEL — make a subset topology library	45
2.2.12	WRITE_TOPOLOGY_MODEL — write residue topology library	46
2.3	Handling of atomic coordinates	47
2.4	Comparison and searching of sequences and structures	48
2.4.1	Alignment file format	48
2.4.2	READ_ALIGNMENT — read sequences and/or their alignment	50
2.4.3	READ_ALIGNMENT2 — read 2nd alignment	51
2.4.4	CHECK_ALIGNMENT — check alignment for modeling	51
2.4.5	COLOR_ALN_MODEL — color MODEL according to alignment	52
2.4.6	COMPARE_ALIGNMENTS — compare two alignments	53
2.4.7	SEQUENCE_TO_ALI — copy MODEL sequence and coordinates to alignment	53
2.4.8	WRITE_ALIGNMENT — write sequences and/or their alignment	54
2.4.9	DESCRIBE — describe proteins	55
2.4.10	ID_TABLE — calculate percentage sequence identities	56
2.4.11	SEQUENCE_COMPARISON — compare sequences in alignment	56
2.4.12	DENDROGRAM — clustering	57
2.4.13	PRINCIPAL_COMPONENTS — clustering	57
2.4.14	ALIGN — align two (blocks of) sequences	58
2.4.15	ALIGN2D — align sequences with structures	59
2.4.16	MALIGN — align two or more sequences	61
2.4.17	ALIGN_CONSENSUS — consensus sequence alignment	62
2.4.18	SUPERPOSE — superpose MODEL2 on MODEL given alignment	63
2.4.19	COMPARE — compare 3D structures given alignment	65
2.4.20	ALIGN3D — align two structures	66
2.4.21	MALIGN3D — align two or more structures	68
2.4.22	EXPAND_ALIGNMENT — put all models into alignment	70
2.4.23	SEQUENCE_SEARCH — search for similar sequences	70
2.4.24	DELETE_ALIGNMENT — delete alignment	72
2.4.25	SEGMENT_MATCHING — align segments	73
2.5	Calculation of spatial restraints	75

2.5.1	Specification of restraints	75
2.5.2	Specification of pseudo atoms	76
2.5.3	MAKE_RESTRAINTS — make restraints	80
2.5.4	DEFINE_SYMMETRY — define similar segments	84
2.5.5	PICK_RESTRAINTS — pick restraints for selected atoms	86
2.5.6	CONDENSE_RESTRAINTS — remove unselected restraints	87
2.5.7	ADD_RESTRAINT — add restraint	88
2.5.8	DELETE_RESTRAINT — unselect restraint	89
2.5.9	REINDEX_RESTRAINTS — renumber MODEL2 restraints for MODEL	89
2.5.10	SPLINE_RESTRAINTS — approximate restraints by splines	90
2.5.11	READ_RESTRAINTS — read spatial restraints	90
2.5.12	WRITE_RESTRAINTS — write spatial restraints	91
2.6	Optimization of the model	92
2.6.1	MAKE_SCHEDULE — create optimization schedule	92
2.6.2	READ_SCHEDULE — read optimization schedule	93
2.6.3	WRITE_SCHEDULE — write optimization schedule	93
2.6.4	ENERGY — evaluate MODEL given restraints	94
2.6.5	OPTIMIZE — optimize MODEL given restraints	96
2.6.6	SWITCH_TRACE — open new optimization trace file	100
2.6.7	DEBUG_FUNCTION — test code self-consistency	101
3	MODELLER scripts	103
3.1	Flowchart of comparative modeling by MODELLER	103
3.2	Script for comparative modeling	105
3.3	Script for modeling of loops	108
4	TOP, MODELLER scripting language	111
5	Methods	113
5.1	Dynamic programming for sequence and structure comparison and searching	113
5.1.1	Pairwise comparison	113
5.1.2	Variable gap penalty	114
5.1.3	Local <i>versus</i> global alignment	114
5.1.4	Similarity <i>versus</i> distance scores	115
5.1.5	Multiple comparisons	115
5.2	Optimization of the objective function by MODELLER	115
5.2.1	Function	115
5.2.2	Optimizers	116
5.3	Equations used in the derivation of the molecular pdf	117
5.3.1	Features and their derivatives	117
5.3.2	Restraints and their derivatives	119
5.4	List of commands, arguments, and default values	123

List of Figures

1.1	<i>Comparative protein modeling by satisfaction of spatial restraints.</i>	10
1.2	<i>Sample spatial restraint.</i>	11
1.3	<i>Optimization of the objective function.</i>	11

List of Tables

2.1	<i>List of file types.</i>	37
2.2	<i>List of mathematical forms of restraints.</i>	78
2.3	<i>List of feature types that can be restrained.</i>	79
2.4	<i>List of “physical” restraint types.</i>	79
2.5	<i>Columns in an optimization trace file.</i>	101
3.1	<i>List of MODELLER scripts.</i>	103

Copyright notice

MODELLER, a protein structure modeling program.
Copyright © 1989–2000 Andrej Šali.

This program is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for any purpose. The entire risk as to the quality and performance of the program is with you.

Distribution of the program is allowed only with the author's written consent.

Acknowledgments

I am grateful to my PhD supervisor Professor Tom L. Blundell in whose laboratory at Birkbeck College the program was initiated.

I would also like to thank Professor Martin Karplus who allowed some of the data in the CHARMM topology and library files to be used with MODELLER.

I am in debt to the MODELLER users for their constructive criticisms and suggestions.

MODELLER was written when at

1989–1990: Department of Crystallography, Birkbeck College
University of London, Malet St, London WC1E 7HX, UK.

1990–1991: ICRF Unit of Structural Molecular Biology, Birkbeck College
Malet St, London WC1E 7HX, UK.

1991–1994: Department of Chemistry, Harvard University
12 Oxford St, Cambridge, MA 02138, USA.

1995–: The Rockefeller University,
1230 York Ave, New York, NY 10021, USA.

Chapter 1

Introduction

1.1 What is MODELLER?

MODELLER is a computer program that models 3D structure of proteins by satisfaction of spatial restraints.

MODELLER is most frequently used for homology or comparative protein structure modeling: The user provides an alignment of a sequence to be modeled with known related structures and MODELLER will automatically calculate a model with all non-hydrogen atoms.

More generally, the input to the program are restraints on the spatial structure of the amino acid sequence(s) and ligands to be modeled. The output is a 3D structure that satisfies these restraints as well as possible. Restraints can in principle be derived from a number of different sources. These include related protein structures (comparative modeling), NMR experiments (NMR refinement), rules of secondary structure packing (combinatorial modeling), cross-linking experiments, fluorescence spectroscopy, image reconstruction in electron microscopy, site-directed mutagenesis, intuition, residue-residue and atom-atom potentials of mean force, *etc.* The restraints can operate on distances, angles, dihedral angles, pairs of dihedral angles and some other spatial features defined by atoms or pseudo atoms. Presently, MODELLER automatically derives the restraints only from the known related structures and their alignment with the target sequence.

A 3D model is obtained by optimization of a molecular probability density function (pdf). The molecular pdf for comparative modeling is optimized with the variable target function procedure in Cartesian space that employs methods of conjugate gradients and molecular dynamics with simulated annealing.

MODELLER can also perform multiple comparison of protein sequences and/or structures, clustering of proteins, and searching of sequence databases. The program is used with a scripting language and does not include any graphics. It is written in standard FORTRAN 90 and is meant to run on a UNIX computer.

1.2 MODELLER bibliography

More information about MODELLER and its applications can be found in the following sources:

1. Šali, A. and Blundell, T. L. Definition of general topological equivalence in protein structures: A procedure involving comparison of properties and relationships through simulated annealing and dynamic programming. *J. Mol. Biol.* **212**, 403–428 (1990).
2. Šali, A., Overington, J. P., Johnson, M. S., and Blundell, T. L. From comparisons of protein sequences and structures to protein modelling and design. *TIBS* **15**, 235–240 (1990).
3. Johnson, M. S., Šali, A., and Blundell, T. L. Phylogenetic relationships from three-dimensional protein structures. *Meth. Enzymol.* **183**, 670–690 (1990).
4. Johnson, M. S., Overington, J. P., and Šali, A. Knowledge-based protein modelling: Human plasma kallikrein and human neutrophil defensin. In *Current Research in Protein Chemistry: Techniques Structure and Function*, Villafranca, J. J., editor, 567–574. Academic Press, Inc., San Diego (1990).
5. Šali, A. *Modelling three-dimensional structure of proteins from their sequence of amino acid residues*. PhD thesis, University of London, London, (1991).
6. Šali, A. and Blundell, T. L. Comparative protein modelling by satisfaction of spatial restraints. *J. Mol. Biol.* **234**, 779–815 (1993).
7. Šali, A., Matsumoto, R., McNeil, H. P., Karplus, M., and Stevens, R. L. Three-dimensional models of four mouse mast cell chymases. Identification of proteoglycan-binding regions and protease-specific antigenic epitopes. *J. Biol. Chem.* **268**, 9023–9034 (1993).
8. Šali, A. and Blundell, T. L. Comparative protein modelling by satisfaction of spatial restraints. In *Protein Structure by Distance Analysis*, Bohr, H. and Brunak, S., editors, 64–86. IOS Press, Amsterdam (1994).
9. Šali, A. and Overington, J. Derivation of rules for comparative protein modeling from a database of protein structure alignments. *Protein Sci.* **3**, 1582–1596 (1994).
10. Šali, A. MODELER: Implementing 3D protein modeling. In *mc²*, volume 2, 5. Molecular Simulations Inc., Burlington, MA (1995).
11. Matsumoto, R., Šali, A., Ghildyal, N., Karplus, M., and Stevens, R. L. Packaging of proteases and proteoglycans in the granules of mast cells and other hematopoietic cells. A cluster of histidines in mouse mast cell protease-7 regulates its binding to heparin serglycin proteoglycan. *J. Biol. Chem.* **270**, 19524–19531 (1995).
12. Šali, A. Protein modeling by satisfaction of spatial restraints. *Molecular Medicine Today* **1**, 270–277 (1995).
13. Šali, A., Potterton, L., Yuan, F., van Vlijmen, H., and Karplus, M. Evaluation of comparative protein structure modeling by MODELLER. *Proteins* **23**, 318–326 (1995).
14. Šali, A. Modelling mutations and homologous proteins. *Curr. Opin. Biotech.* **6**, 437–451 (1995).
15. Sheng, Y., Šali, A., Herzog, H., Lahnstein, J., and Krilis, S. Modelling, expression and site-directed mutagenesis of human β_2 -glycoprotein I: Identification of the major phospholipid binding site. *J. Immunol.* **157**, 3744–3751 (1996).
16. Ghildyal, N., Friend, D. S., Stevens, R. L., Austen, K. F., Huang, C., Penrose, J., Šali, A., and Gurish, M. F. Fate of two mast cell tryptases following passive systemic anaphylaxis of BALB/c and V3 mastocytosis mice. Prolonged retention of exocytosed mMCP-6 in connective tissues and rapid accumulation of enzymatically active mMCP-7 in the blood chymases. *J. Exp. Med.* **184**, 1061–1073 (1996).
17. Xu, L. Z., Sánchez, R., Šali, A., and Heintz, N. Ligand specificity of brain lipid binding protein. *J. Biol. Chem.* **271**, 24711–24719 (1996).

18. Wu, S., de Lencastre, H., Šali, A., and Tomasz, A. A phosphoglucosyltransferase-like gene essential for the optimal expression of methicillin resistance in *Staphylococcus aureus*: Molecular cloning and DNA sequencing. *Microbial Drug Resistance* **2**, 277–286 (1996).
19. Huang, C., Wong, G. W., Ghildyal, N., Gurish, M. F., Šali, A., Matsumoto, R., Qiu, W.-T., and Stevens, R. L. The tryptase, mouse mast cell protease 7, exhibits anticoagulant activity *in vivo* and *in vitro* due to its ability to degrade fibrinogen in the presence of the diverse array of protease inhibitors in plasma. *J. Biol. Chem.* **272**, 31885–31893 (1997).
20. Sánchez, R. and Šali, A. Comparative protein modeling as an optimization problem. *Journal of Molecular Structure (Theochem)* **398**, 489–496 (1997).
21. Sánchez, R. and Šali, A. Advances in comparative protein-structure modeling. *Curr. Opin. Struct. Biol.* **7**, 206–214 (1997).
22. Koulich, D., Orlova, M., Malhotra, A., Šali, A., Darst, S. A., and Borukhov, S. Domain organization of *Escherichia coli* transcript cleavage factors GreA and GreB. *J. Biol. Chem.* **272**, 7201–7210 (1997).
23. Hunt, J. E., Friend, D. S., Gurish, M. F., Feyfant, E., Šali, A., Huang, C., Ghildyal, N., Stechshulte, S., Austen, K. F., and Stevens, R. L. Mouse mast cell protease (mMCP) 9, a novel member of the chromosome 14 family of serine proteases that is selectively expressed in uterine mast cells. *J. Biol. Chem.* **46**, 29158–29166 (1997).
24. Sánchez, R. and Šali, A. Evaluation of comparative protein structure modeling by MODELLER-3. *Proteins Suppl.* **1**, 50–58 (1997).
25. Guenther, B., Onrust, R., Šali, A., O'Donnell, M., and Kuriyan, J. Crystal structure of the δ' subunit of the clamp-loader complex of *E. coli* DNA polymerase III. *Cell* **91**, 335–345 (1997).
26. Sánchez, R., Badretdinov, A. Y., Feyfant, E., and Šali, A. Homology protein structure modeling. *Transactions Amer. Cryst. Assoc.* **32**, 81–91 (1997).
27. Sánchez, R. and Šali, A. Comparative protein structure modeling: Introduction and practical examples with MODELLER. In *Protein Structure Prediction: Methods and Protocols*, Webster, D. M., editor, 97–129. Humana Press (2000).
28. Huang, C., Šali, A., and Stevens, R. L. Regulation and function of mast cell proteases in inflammation. *J. Clin. Immunol.* **18**, 169–183 (1998).
29. Kandiah, D. A., Šali, A., Sheng, Y., Victoria, E. J., Marquis, D. M., Coutts, S. M., and Krilis, S. Current insights into the antiphospholipid syndrome: Clinical, immunological and molecular aspects. *Adv. Immunol.* **70**, 507–563 (1998).
30. Wolf, E., Vassilev, A., Makino, Y., Šali, A., Nakatani, Y., and Burley, S. K. Crystal structure of a GCN5-related N-acetyltransferase: *Serratia marcescens* aminoglycoside 3-N-acetyltransferase. *Cell* **94**, 51–61 (1998).
31. Sánchez, R. and Šali, A. Large-scale protein structure modeling of the *Saccharomyces cerevisiae* genome. *Proc. Natl. Acad. Sci. USA* **95**, 13597–13602 (1998).
32. Šali, A. 100,000 protein structures for the biologist. *Nat. Struct. Biol.* **5**, 1029–1032 (1998).
33. Sánchez, R. and Šali, A. MODBASE : A database of comparative protein structure models. *Bioinformatics* **15**, 1060–1061 (1999).
34. Fiser, A., Sánchez, R., Melo, F., and Šali, A. Comparative protein structure modeling. In *Computational Biochemistry and Biophysics, in press*, Watanabe, M., Roux, B., MacKerell, A., and Becker, O., editors. Marcel Dekker (2000).
35. Sánchez, R. and Šali, A. Comparative protein structure modeling in genomics. *J. Comp. Phys.* **151**, 388–401 (1999).
36. Wu, G., Fiser, A., ter Kuile, B., Šali, A., and Müller, M. Convergent evolution of *Trichomonas vaginalis* lactate dehydrogenase from malate dehydrogenase. *Proc. Natl. Acad. Sci. USA* **96**, 6285–6290 (1999).

37. Nagata, T., Gupta, V., Kim, W.-Y., Šali, A., Chait, B. T., Shigesada, K., Ito, Y., and Werner, M. H. Immunoglobulin motif DNA recognition and heterodimerization for the PEBP2/CBF Runt-domain. *Nat. Str. Biol.* **6**, 615–619 (1999).
38. Miwa, J. M., Ibanez-Tallon, I., Crabtree, G. W., Sánchez, R., Šali, A., Role, L. W., and Heintz, N. lynx1, an endogenous toxin-like modulator of nicotinic acetylcholine receptors in the mammalian CNS. *Neuron* **23**, 105–114 (1999).
39. Huang, C., Morales, G., Vagi, A., Chanasyk, K., Ferrazzi, M., Burklow, C., Qiu, W. T., Feyfant, E., Šali, A., and Stevens, R. L. Formation of enzymatically active, homotypic and heterotypic tetramers of mouse mast cell tryptases. *J. Biol. Chem.* **275**, 351–358 (2000).
40. Wong, G. W., Tang, Y., Feyfant, E., Šali, A., Huang, C., and Stevens, R. L. Identification of a new tryptase in mice and humans which possesses a novel C-terminal, membrane spanning segment. *J. Biol. Chem.* **274**, 30784–30793 (1999).
41. Wu, G., Morrison, H. G., Fiser, A., McArthur, A. G., Šali, A., Sogin, M. L., and Müller, M. Core histones of the amitochondriate protist, *Giardia lamblia*. *Mol. Biol. Evol.* **17**, 1156–1163 (2000).
42. Burley, S. K., Almo, S. C., Bonanno, J. B., , Capel, M., Chance, M. R., Gaasterland, T., Lin, D., Šali, A., Studier, F. W., and Swaminathan, S. Structural genomics: beyond the Human Genome Project. *Nat. Genet.* **23**, 151–157 (1999).
43. Martí-Renom, M. A., Stuart, A., Fiser, A., Sánchez, R., Melo, F., and Šali, A. Comparative protein structure modeling of genes and genomes. *Ann. Rev. Biophys. Biomolec. Struct.* **29**, 291–325 (2000).
44. Sánchez, R., Pieper, U., Mirković, N., de Bakker, P. I. W., Wittenstein, E., and Šali, A. MODBASE, a database of annotated comparative protein structure models. *Nucl. Acids Res.* **28**, 250–253 (2000).
45. Šali, A. and Kuriyan, J. Challenges at the frontiers of structural biology. *Trends Biochem. Sci.* **22**, M20–M24 (1999).
46. Pieper, U., Eswar, N., Sánchez, R., Mirković, N., Lane, W., Sammut, M., John, B., and Šali, A. MODBASE, a database of annotated comparative protein structure models. *Nucl. Acids Res.* , in press (2001).
47. Groft, C. M., Beckmann, R., Šali, A., and Burley, S. K. Crystal structures of ribosome anti-association factor IF6. *Nat. Struct. Biol.* **7**, 1156–1164 (2000).
48. Sánchez, R., Pieper, U., Melo, F., Eswar, N., Martí-Renom, M., Madhusudhan, M., Mirković, N., and Šali, A. Protein structure modeling for structural genomics. *Nat. Struct. Biol.* **7**, 986–990 (2000).
49. Fiser, A., Do, R. K. G., and Šali, A. Modeling of loops in protein structures. *Protein Science* **9**, 1753–1773 (2000).

Please quote A. Šali and T. L. Blundell, “Comparative protein modelling by satisfaction of spatial restraints” *J. Mol. Biol.* **234**, 779–815 (1993) in your publications using MODELLER.

I will greatly appreciate it you send me copies of any papers using MODELLER to

Andrej Šali
The Rockefeller University
1230 York Avenue
New York, NY 10021-6399
USA

tel +1 212 327 7550
fax +1 212 327 7540
email sali@rockefeller.edu
URL <http://guitar.rockefeller.edu/>

1.3 Distribution

MODELLER is available free of charge to academic non-profit institutions.

First, please use the MODELLER home page at <http://guitar.rockefeller.edu/modeller/modeller.html> or the anonymous ftp account, in directory pub/modeller, on guitar.rockefeller.edu (IP 129.85.13.195) to obtain at least (`modeller6a-data.tar.Z`), and an executable for each machine type that you want to use (`mod6a.*`) (see file `INSTALLATION`). Next, please access the academic license server from the MODELLER home page to apply for the MODELLER key that has to be assigned to the environment variable `KEY_MODELLER6` in your login script (`.cshrc`). See file `INSTALLATION` for installation instructions.

There is a MODELLER users email list. You can access it from the MODELLER home page or subscribe to it directly by sending an email message with the word “subscribe” in its body to `modeller_usage@guitar.rockefeller.edu`.

A graphical interface to MODELLER is available as part of QUANTA, INSIGHTII, and WEBLAB GENEEXPLORER, interactive molecular modeling programs from MSI, San Diego, with many tools for protein modeling and structural analysis. QUANTA, INSIGHTII and GENEEXPLORER facilitate preparation of input files for MODELLER (*e.g.*, an alignment file) as well as an analysis of results (*e.g.*, an evaluation of the models). If you are interested in these programs, please contact

Dr. David Edwards
Molecular Simulations Inc.
9685 Scranton Road
San Diego, CA 92121-3752
USA
tel +1-858-799-5374
fax +1-858-458-0136
email dje@msi.com
URL <http://www.msi.com/>

1.4 Installation

The following installation instructions are from the INSTALLATION file in the root directory of the MODELLER distribution. See Section 1.3 for how to obtain MODELLER.

INSTALLATION

M O D E L L E R 6v0

Copyright(c) 1989-2001 Andrej Sali
All Rights Reserved

** PLATFORMS

MODELLER is written in Fortran 90 and runs on Silicon Graphics under IRIX, on Sun workstation under Solaris, IBM under AIX, on DEC Alphas and Pentium PC's under Linux.

** INSTALLATION

See file modeller.README for information about how to get MODELLER. The source code is not generally available. Hence, most users are limited to the compiled versions of MODELLER. Program is distributed as a single compressed tar archive (modeller6v0.tar.Z) that contains scripts, libraries, examples, documentation (in PDF and HTML formats) and executables for different platforms and operating systems.

- 1) Download modeller6v0.tar.Z file into the desired directory on your computer.
- 2) Unpack the file:

```
uncompress modeller6v0.tar.Z
tar xvf modeller6v0.tar
```

The result of unpacking will be directory ./modeller6v0 (next to the modeller6v0.tar.Z file) with the following uncompressed files and directories:

doc/	MODELLER documentation directory
examples/	directory containing examples and tutorials
Install	installation script
INSTALLATION	this file
modeller.README	file describing distribution and registration
modlib/	libraries and data files for the program
scripts/	script files used to compile and use MODELLER
src/	directory with executables

- 3) Go to the ./modeller6v0 directory and run the installation script

```
./Install
```

Answer several questions as prompted. If you make a mistake, you can re-run the script.

- 4) If the script installation was successful you may stop here. Otherwise you have to install MODELLER manually as described below.

- 5) Include into your login script the lines below. Edit them appropriately to suit your system. In case of csh or tcsh the login script is ".cshrc" (you can generally find your current shell by typing 'echo \$shell').

```
#####
```

```
# this is a location of the modeller6v0 directory
# (i.e where you unpacked tar.Z file)
setenv MODINSTALL6v0 /usr/local/modeller/modeller6v0
```

```
# please comment out those that don't apply to you:
setenv EXECUTABLE_TYPE6v0 i386-absoft # for Linux
setenv EXECUTABLE_TYPE6v0 iris4d     # for SGI
setenv EXECUTABLE_TYPE6v0 sun4       # for SUN
setenv EXECUTABLE_TYPE6v0 alpha      # for DEC Alpha
setenv EXECUTABLE_TYPE6v0 rs6000     # for IBM
```

```
setenv LIBS_LIB6v0
$MODINSTALL6v0/modlib/libs.lib
```

```
# replace KEY_MODELLER with the Modeller key you obtained
# after signing the online license agreement
setenv KEY_MODELLER6v0
```

```
set path=($path $MODINSTALL6v0/bin)
limit stacksize unlimited
```

```
#####
```

- 6) Source the login script by executing the command:

```
source ~/.cshrc
```

- 5) Install the program:

```
a) change directory:
   cd $MODINSTALL6v0/src/main
```

b) execute command:
make install

For additional information visit our web site:
<http://guitar.rockefeller.edu>

Sincerely,
Modeller Team
December, 2001.

1.5 Bug reports

Please report MODELLER bugs by e-mail to the MODELLER users list at `modeller_usage@guitar.rockefeller.edu`. It is best if you e-mail an uuencoded compressed tar archive of the directory with all input and output files:

```
tar cvf name.tar ./directory_name
compress name.tar
uuencode name.tar.Z name.tar.Z > name.uue
mail -s 'MODELLER bug' modeller_usage@guitar.rockefeller.edu < name.uue
```

If the problem is not apparent in the log files, please include in the archive a README file with a short description of the problem.

1.6 Method for comparative protein structure modeling by MODELLER

MODELLER implements an automated approach to comparative protein structure modeling by satisfaction of spatial restraints (Figure 1.1) [?]. The method and its applications to biological problems are described in detail in references listed in Section 1.2. Briefly, the core modeling procedure begins with an alignment of the sequence to be modeled (target) with related known 3D structures (templates). This alignment is usually the input to the program. The output is a 3D model for the target sequence containing all mainchain and sidechain non-hydrogen atoms. Given an alignment, the model is obtained without any user intervention. First, many distance and dihedral angle restraints on the target sequence are calculated from its alignment with template 3D structures (Figure 1.2). The form of these restraints was obtained from a statistical analysis of the relationships between many pairs of homologous structures. This analysis relied on a database of 105 family alignments that included 416 proteins with known 3D structure [?]. By scanning the database, tables quantifying various correlations were obtained, such as the correlations between two equivalent $C_{\alpha} - C_{\alpha}$ distances, or between equivalent mainchain dihedral angles from two related proteins. These relationships were expressed as conditional probability density functions (pdf's) and can be used directly as spatial restraints. For example, probabilities for different values of the mainchain dihedral angles are calculated from the type of a residue considered, from mainchain conformation of an equivalent residue, and from sequence similarity between the two proteins. Another example is the pdf for a certain $C_{\alpha} - C_{\alpha}$ distance given equivalent distances in two related protein structures (Figure 1.2). An important feature of the method is that the spatial restraints are obtained empirically, from a database of protein structure alignments. Next, the spatial restraints and CHARMM energy terms enforcing proper stereochemistry [?] are combined into an objective function. Finally, the model is obtained by optimizing the objective function in Cartesian space. The optimization is carried out by the use of the variable target function method [?] employing methods of conjugate gradients and molecular dynamics with simulated annealing (Figure 1.3). Several slightly different models can be calculated by varying the initial structure. The variability among these models can be used to estimate the errors in the corresponding regions of the fold.

There are additional specialized modeling protocols, such as that for the modeling of loops (Section 3.3).

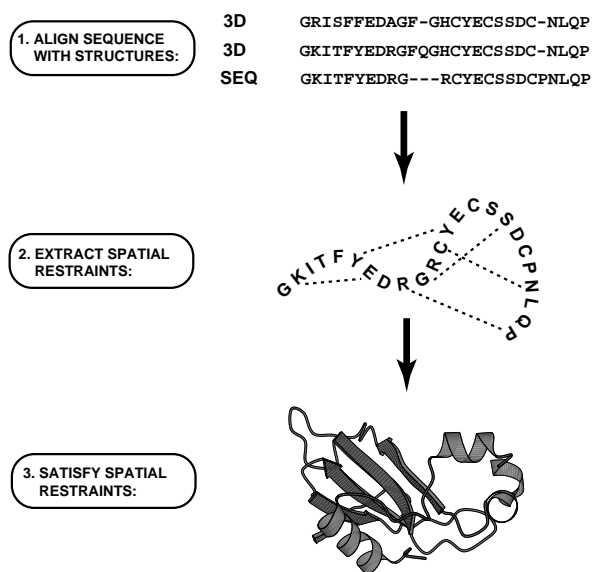


Figure 1.1: *Comparative protein modeling by satisfaction of spatial restraints.* First, the known, template 3D structures ('3D') are aligned with the target sequence to be modeled ('SEQ') Second, spatial features, such as $C_{\alpha} - C_{\alpha}$ distances, hydrogen bonds, and mainchain and sidechain dihedral angles, are transferred from the templates to the target. Thus, a number of spatial restraints on its structure are obtained. Third, the 3D model is obtained by satisfying all the restraints as well as possible.

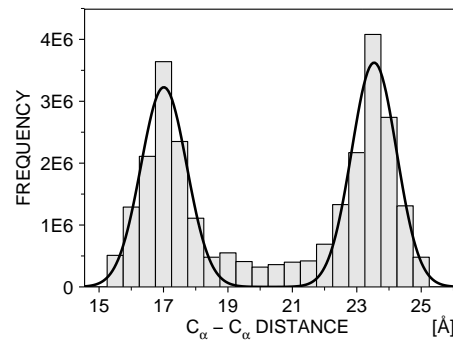


Figure 1.2: *Sample spatial restraint.* A restraint on a given C_α - C_α distance, d , is expressed as a conditional probability density function that depends on two other equivalent distances ($d' = 17.0$ and $d'' = 23.5$): $p(d/d', d'')$. The restraint (continuous line) is obtained by least-squares fitting a sum of two Gaussian functions to the histogram, which in turn is derived from the database of alignments of protein structures. In practice, more complicated restraints are used that depend on additional information, such as similarity between the proteins, solvent accessibility, and distance from a gap in the alignment [?].

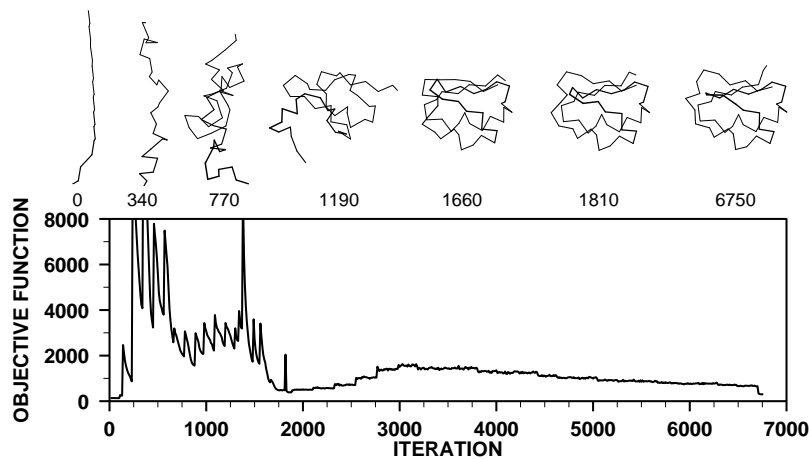


Figure 1.3: *Optimization of the objective function.* Optimization of the objective function (curve) starts with a distorted average of template structures (not with an extended structure as shown here). The iteration number is indicated below each sample structure. In this run, the first $\sim 2,000$ iterations correspond to the variable target function method relying on the conjugate gradients technique. This approach first satisfies sequentially local restraints and slowly introduces longer range restraints until the complete objective function is optimized. In the last 4,750 iterations for this model, molecular dynamics with simulated annealing is used to refine the model. Typically, a model is calculated in the order of minutes on a PC workstation.

1.7 Comparative protein modeling primer

This section outlines all the stages in a comprehensive comparative modeling session. In contrast, the tutorial section (Section 1.8) describes a hands-on example illustrating a very simple modeling case. Many “frequently-asked-questions” (FAQ) are answered in Section 1.9. An introductory, hands-on primer to comparative modeling with several MODELLER examples can be found in ?. With a convenient structure display and manipulation program, it should be possible to do even the difficult modeling cases in only a few days. To find sample script files for all the MODELLER tasks discussed in this section, read file `examples/all-steps/README`. To obtain additional examples of TOP scripts using a certain command, explore the `examples` directory, especially the `examples/commands` sub-directory.

1.7.1 Finding structures and sequences related to the target sequence

Before any modeling can begin, the sequences and segments with known 3D structures that are related to the sequence being modeled must be found. This can be achieved by the MODELLER **SEQUENCE_SEARCH** command. The search relies on a database of structures that are representative [?] of the whole Protein Data Bank (PDB) [?, ?]. The PDB codes of these representative structures (about 3,000 codes) are listed in file `modlib/CHAINS_3.0_40_XN.cod` and their sequences are stored in file `modlib/CHAINS_all.seq`, which includes approximately 16,000 sequences for all the unique non-model chains in PDB longer than 25 amino acid residues. The representative structures are likely to have less than 40% sequence identity to each other and the length difference that is at least 30% of the shorter chain or 30 amino acid residues, whichever is smaller. The codes of other known PDB structures related to the representative structures at >40% sequence identity are listed in file `modlib/CHAINS_3.0_40_XN.grp`. A sample TOP script for searching by **SEQUENCE_SEARCH** is in `examples/all-steps/search.top`. Sequences related to the target are identified by their Z-scores that are larger than 4 or 5 (log file column SIGNIF).

For more difficult modeling problems when **SEQUENCE_SEARCH** does not find any homologs, template matching or threading methods can be used. Widely used programs for threading include PROFIT [?], THREADER [?], and the Web server of the David Eisenberg group at UCLA (<http://www.mbi.ucla.edu/people/frsvr/frsvr>).

It may be beneficial to identify related sequences without known 3D structures at this stage. This is most conveniently achieved by PSI-BLAST [?]. Using as many sequences as possible may improve the quality of the alignment prepared in the next two stages.

1.7.2 Preparing an initial family alignment of all structures and sequences

The second stage is to prepare a multiple alignment of all the structures and sequences in the family of interest. An initial structural alignment of the structures can be obtained by the MODELLER’s **MALIGN3D** command. A very robust protein structure alignment program is CE [?]. It is usually better to include all the structures related to the representative PDB structures, as listed in the `modlib/CHAINS_3.0_40_XN.grp` file. Using the **ALIGN2D**, **ALIGN** or **MALIGN** commands, the multiple structural alignment can then be aligned as one block with all the related sequences. Unfortunately, in most cases, automatically derived alignments have to be manually edited to optimize the quality of the model derived from the alignment. This is the task for the next stage.

1.7.3 Becoming familiar with the family fold and improving the alignment

This is a very important stage because it may result in a significantly improved model. It mostly involves visual inspection of the superposed structures on a graphics terminal, using a program such as RASMOL (Roger Sayle, Glaxo, <http://www.umass.edu/microbio/rasmol/>), QUANTA or INSIGHTII. However, the **CHECK_ALIGNMENT** command of MODELLER should also be used. You can obtain files with multiply superposed structures by using the **MALIGN3D** command with **WRITE_FIT** set to on (you can use any alignment for this purpose). The aim at this stage is to study the family fold, to establish the relationships between various members of the family, and to determine which regions are more conserved and which regions are more variable. This information is used to improve the initial automatically derived alignment. For example, if necessary, gaps are removed from helices and strands; they should be moved into those exposed regions that show large variations in the family of known structures and to the tips of loops. The role of disulfides and *cis*-prolines, if any, is noted. MODELLER will try to

deal with those two features automatically but it is prudent to be careful. It has to be decided whether or not to build models for multi-subunit assemblies and whether or not to include various ligands, such as water molecules, cofactors, metal ions, inhibitors, or substrates. QUANTA and INSIGHTII have a set of tools that facilitate inspection and editing of multiple alignments.

In order to obtain the best possible model, it is very important to understand how the alignment is used by MODELLER [?]. In outline, for the aligned regions, MODELLER tries to derive a 3D model for the target sequence that is as close to one or the other of the template structures as possible while also satisfying stereochemical restraints (*e.g.*, bond lengths, angles, non-bonded atom contacts, . . .); the inserted regions, which do not have any equivalent segments in any of the templates, are modeled in the context of the whole molecule, but using their sequence alone. This way of deriving a model means that whenever a user aligns a target residue with a template residue, he tells MODELLER to treat the aligned residues as **structurally** equivalent.

1.7.4 Selecting the templates

The new improved alignment is input to the **ID_TABLE** or **COMPARE_SEQUENCES** commands to construct a matrix of pairwise sequence distances. This matrix is then used either to prepare an ‘evolutionary’ tree for the whole family or to cluster the proteins by the principal components technique available through the **PRINCIPAL_COMPONENTS** command of MODELLER. For evolutionary trees, the **DENDROGRAM** command of MODELLER or the PHYLIP program written by Joe Felsenstein can be used (you can get PHYLIP by anonymous FTP from evolution.genetics.washington.edu/pub/phylip) [?]. The clustering is then examined to decide which known structures are suitable templates for model building in the next stage. Usually, all significantly different structures in the cluster that contains the target sequence are used. It is not always best to use all related 3D structures as templates because the objective function may become too rugged, sometimes resulting in sub-optimal solutions (*e.g.*, six templates is a large number of templates). It also does not make sense to include two relatively similar templates solved at a high and low resolution; use only the high resolution template. Depending on the modeling problem at hand, other factors can be considered in the selection of templates, such as ligands bound to the template and/or target, whether the template structure was solved in solution or in a crystal, *etc.* Moreover, more experienced users can try to use a smaller number of templates for mainchain distance restraints and a larger number of templates for sidechain conformation, but that involves editing the TOP scripts for comparative modeling. Also, templates can be very short, such as loops from unrelated protein structures that fit on the given framework regions; for example, canonical loops [?] could be used as templates in modeling a complementarity determining region of an immunoglobulin. However, MODELLER’s *ab initio* loop modeling facility is the preferred way to model loops (Section 3.3).

1.7.5 Model building

The alignment and the list of templates are used by MODELLER to derive several slightly different models automatically. This stage is straightforward and is described in the tutorial (Section 1.8). However, do check the log file for error messages by searching for the ‘_E>’ string. Usually, the representative model is that which has the lowest value of the molecular pdf. If the models are constructed by the standard procedure, this value is reported in the log file as well as in the REMARK record of the output PDB files with the models.

1.7.6 Evaluating the models

The model is evaluated internally and externally. The internal self-consistency check is that the model has to satisfy most restraints used to calculate it, especially the stereochemical restraints. If some restraints are grossly violated in all the models it is likely that the alignment in the corresponding region is incorrect. The restraint violations are reported by the **ENERGY** command and can be found at the end of the log file. External tests include programs such as PROCHECK by Roman Laskowski and Janet Thornton (anonymous FTP at [ftp.biochem.ucl.ac.uk](ftp://ftp.biochem.ucl.ac.uk)) [?], the EE1 web server by Themis Lazaridis (<http://mingus.sci.ccnycuny.edu/server/>) and various 3D profile tests including the PROFILE3D program written in the David Eisenberg group (e-mail david@uclaue.mbi.ucla.edu) [?] and PROSII written in the Manfred Sippl group (anonymous FTP at <gundi.came.sbg.ac.at>) [?]. QUANTA and INSIGHTII also offer various test options.

It is useful to compare the models among themselves because those regions that are most variable are also likely

to be most in error. Another useful comparison is between the representative model and the templates. Start by comparing the C_α traces, then continue with the backbone comparison and finally include the sidechains.

The aim of evaluations is to determine whether or not the model is acceptable. If it is not acceptable, that is if the current model violates some restraints, fails the profile tests, or simply does not appear satisfactory, these evaluations should help to re-align the target sequence and the templates for the next cycle of modeling. It may also be that the model violates restraints because the optimizer did not find a good optimum of the objective function. In such a case, you could make the optimization more thorough using options to the 'model' routine (Section 3.2).

1.7.7 Repeat the cycle

The cycle of template selection, alignment, modeling, and evaluation should be repeated until the model is good enough or until no further improvement is possible.

1.8 Tutorial on using MODELLER for comparative modeling

This section is a ‘hands on’ description of the most basic use of MODELLER in comparative modeling. For an outline of the main stages in comparative modeling, see Section 1.7. For “frequently-asked-questions” (FAQ), see Section 1.9. An introductory, hands-on primer to comparative modeling with several MODELLER examples can be found in ?.

The input are Protein Data Bank (PDB) atom files of known protein structures and their alignment with the target sequence to be modeled. The output is a model for the target that includes all non-hydrogen atoms. Although MODELLER can calculate sequence and structure alignments, it is better to prepare the alignment carefully by other means. The alignment can also contain very short segments such as loops, secondary structure motifs, *etc.*

This tutorial assumes that MODELLER is already installed on your computer and that appropriate changes have been made to your login script to install you as a MODELLER user. See Section 1.4 for more details on installation (also in the INSTALLATION file in the MODELLER distribution directory).

1.8.1 Preparing input files

The sample input files in this tutorial can be found in the `examples/tutorial-model` directory of the MODELLER distribution.

There are three kinds of input files: Protein Data Bank atom files with coordinates for the template structures, the alignment file with the alignment of the template structures with the target sequence, and the MODELLER command or script file that tells MODELLER what to do.

Atom files

Each atom file is named `code.atm` where `code` is a short protein code, preferably the PDB code; for example, *Peptococcus aerogenes* ferredoxin would be in a file `1fdx.atm`. The code must be used as that protein’s identifier throughout the modeling. The atom sets do not have to be superposed by the user before comparative modeling is done.

Alignment file

One of the formats for the alignment file is related to the PIR database format; this is the preferred format for comparative modeling:

```
C; A sample alignment in the PIR format; used in tutorial
>P1;5fd1
structureX:5fd1:1      : :106  : :ferredoxin:Azotobacter vinelandii: 1.90: 0.19
AFVVTDNCKIKCKYTDCEVEVCPVDCFYEGPNFLVIHPDECIDCALCEPECPAQAIIFSEDEVPEDMQEFIQLNAELA
EVWPNITEKKDPLPDAEDWDGVKGLQLHLER*
>P1;1fdx
sequence:1fdx:1      : :54   : :ferredoxin:Peptococcus aerogenes: 2.00:-1.00
AYVINDSC--IACGACKPECPVNIQGS--IY AIDADSCIDCGSCASVCPVGAPNPED-----
-----*
```

See Section 2.4.1 for a detailed description of the alignment file format and Section 1.7.3 for the meaning of the alignment in MODELLER. Influence of the alignment on the quality of the model cannot be overemphasized. Command `CHECK_ALIGNMENT` can be used to find some trivial alignment mistakes.

Script file

The script file contains commands for MODELLER, in the TOP language (Chapter 4). A sample script file `model-default.top` to produce one model of sequence `1fdx` from the known structure of `5fd1` and from the alignment between the two sequences is

```
# Homology modelling by the MODELLER TOP routine 'model'.
```

```

INCLUDE                # Include the predefined TOP routines

SET OUTPUT_CONTROL = 1 1 1 1 1 # uncomment to produce a large log file
SET ALNFILE = 'alignment.ali'   # alignment filename
SET KNOWNNS = '5fd1'           # codes of the templates
SET SEQUENCE = '1fdx'         # code of the target
SET ATOM_FILES_DIRECTORY = './:../atom_files' # directories for input atom files
SET STARTING_MODEL = 1        # index of the first model
SET ENDING_MODEL = 1          # index of the last model
                             # (determines how many models to calculate)

CALL ROUTINE = 'model'        # do homology modelling

```

See Section 3.2 for information on the model script and its arguments.

1.8.2 Running MODELLER

To run MODELLER with the script file `model-default.top`, execute the following command:

```
mod model-default
```

A number of intermediary files are created as the program proceeds. After about 30 seconds on a Pentium III workstation, the final `1fdx` model is written to file `1fdx.B99990001`. Examine the `model-default.log` file for information about the run. In particular, one should always check the output of the **CHECK_ALIGNMENT** command, which you can find by searching for `'chkaln'`. Also, check for warning and error messages by searching for `'_W>'` and `'_E>'`, respectively. There should be no error messages; most often, there are some warning messages that can usually be ignored.

1.8.3 Fully automated comparative modeling

Fully automated comparative modeling requires only the target sequence and the coordinates of templates. The structural alignment of the known 3D structures and their alignment with the target sequence is derived automatically. However, the single most important factor that determines the quality of a model is the quality of the alignment. If the alignment is incorrect, the model will also be incorrect. **For this reason, the fully automated option for comparative modeling should not be used unless the sequences are so similar that the calculated alignment is likely to be correct (this usually requires more than 50% sequence identity).** Instead, the alignment should be carefully inspected, optimized by hand, and checked by the **CHECK_ALIGNMENT** command before used in modeling (Section 1.7). Moreover, several iterations of alignment and modeling may be necessary in general. See Section 1.7 for an outline of a general comparative modeling case.

The sample input files for fully automated comparative modeling are located in directory `examples/align-model-steps`. The sample TOP file is

```

# A sample TOP file for fully automated comparative modeling

INCLUDE                # include MODELLER routines
SET ATOM_FILES_DIRECTORY = './:../atom_files' # directory with input atom files
SET SEGFILE           = 'alignment.seg'       # input file w/ templates and target
SET KNOWNNS           = '5fd1' '1fdn' '1fdx' '2fxb' # templates' PDB codes
SET SEQUENCE          = '1fdx'               # target code
SET OUTPUT_CONTROL = 1 1 1 1 2
CALL ROUTINE          = 'full_homol'         # get alignment and a model

```

The `alignment.seg` file is

```
>P1;1fdx
```

```
structureX:1fdx:@:@:54:@:ferredoxin:Peptococcus aerogenes: 2.00:-1.00
AYVINDSCIACGACKPECPVNI IQGSIYAIDADSCIDCGSCASVCPVGAPNPED*
>P1;1fdn
structureX:1fdn:@:@:55:@:ferredoxin:Clostrodium acidiurici: 1.84:-1.0
*
>P1;5fd1
structureX:5fd1:@:@:60:@:ferredoxin:Azotobacter vinelandii: 1.90:0.192
*
>P1;1fxd
structureX:1fxd:@:@:58:@:ferredoxin:Desolfovibrio gigas: 1.70:-1.0
*
>P1;2fxb
structureX:2fxb:@:@:60:@:ferredoxin:Bacillus thermoproteolyticus: 2.30:-1.0
*
```

1.8.4 Improved modeling of loops in MODELLER-6

See Section 3.3 for more information.

1.9 Frequently asked questions (FAQ) and examples

Please also check the archive of the Users Mail List at <http://guitar.rockefeller.edu/modeller/modeller.html>.

1. **I do not care about the details of a model, I only want to calculate it very fast to get a quick idea about how it looks or to confirm that my alignment is clearly unreasonable in the structural sense.**

Only one model can be calculated by this routine because the starting structure is not randomized before optimization. Only a very limited amount of the variable target function optimization with conjugate gradients is done. This is usually for a factor of 3 faster than the default procedure. For example, it takes about 17 seconds of CPU time to model a 60-residue protein on an SGI workstation with a R10000-195 processor.

```
# Very fast homology modelling by the MODELLER TOP routine 'model'.

INCLUDE                # Include the predefined TOP routines
SET ALNFILE   = 'alignment.ali'    # alignment filename
SET KNOWNs    = '5fd1'             # codes of the templates
SET SEQUENCE  = '1fdx'             # code of the target
SET ATOM_FILES_DIRECTORY = './:../atom_files' # directories for input atom files
SET STARTING_MODEL = 2
SET ENDING_MODEL = 2

SET OUTPUT_CONTROL = 1 1 1 1 1
# SET OUTPUT = 'LONG'
SET FINAL_MALIGN3D = 1
CALL ROUTINE = 'very_fast'         # prepare for extremely fast optimization

CALL ROUTINE = 'model'             # do homology modelling
```

2. **How can I refine the model in successive steps?**

There is a pre-defined routine 'select_atoms' which selects the atoms to be moved during optimization. By default, the routine selects all atoms, but you can redefine it to select any subset of atoms and then only those atoms will be refined. They will "feel" the presence of other atoms *via* all the static and possibly dynamic restraints that include both selected and un-selected atoms. For example, the script below would refine only atoms in residues 1 and 2 (file 'examples/tutorial-model/model-segment.top'). The difference between this script and the one for loop modeling is that here the selected regions are optimized with the default optimization protocol and the default restraints, which generally include template-derived restraints. In contrast, the loop modeling routine does not use template-dependent restraints, but does a much more thorough optimization.

```
# Homology modelling by the MODELLER TOP routine 'model'.
# Demonstrates how to refine only a part of the model.
#
# You may want to use the more exhaustive "loop" modeling routines instead.
#

INCLUDE                # Include the predefined TOP routines
SET OUTPUT_CONTROL = 1 1 1 1 0

SET ALNFILE   = 'alignment.ali'    # alignment filename
SET KNOWNs    = '5fd1'             # codes of the templates
SET SEQUENCE  = '1fdx'             # code of the target
SET ATOM_FILES_DIRECTORY = './:../atom_files' # directories for input atom files
SET STARTING_MODEL= 3               # index of the first model
SET ENDING_MODEL = 3               # index of the last model
```



```

# (determines how many models to calculate)
SET NONBONDED_SEL_ATOMS = 2      # selected atoms do not feel the neighbourhood

CALL ROUTINE = 'model'          # do homology modelling

SUBROUTINE ROUTINE = 'select_atoms'
  PICK_ATOMS SELECTION_SEGMENT='1:' '2:', SELECTION_SEARCH='segment', ;
  PICK_ATOMS_SET=1, RES_TYPES='all', ATOM_TYPES='all', ;
  SELECTION_FROM='all', SELECTION_STATUS='initialize'

  RETURN
END_SUBROUTINE

```

3. I want to model one or more loops very thoroughly (meaning spending a lot of CPU time, not necessarily modeling more accurately).

Note that loops and insertions are already modeled by the default modeling routine, so you do not have to do anything special to get a model for the insertions. However, if you really want to focus on loops, you can use the new loop modeling routine 'loop' (Section 3.3). The selected regions are optimized independently many times by a thorough molecular dynamics/simulated annealing procedure, using sequence-dependent restraints only, no homology-derived restraints.

```

# Homology modelling by the MODELLER TOP routine 'model'.
# Demonstrates how to refine only a part of the model.
#
# This can be ran with run_clustor model-loop.top, too.
#
# The difference with model-segment is that the loop is
# refined on the basis of sequence alone, in the context
# of the rest of the structure.

INCLUDE                          # Include the predefined TOP routines

SET OUTPUT_CONTROL = 1 1 1 1 1
SET SEQUENCE = '1fdx'            # code of the target
SET LOOP_MODEL = '1fdx.B99990001' # initial model of the target
SET ATOM_FILES_DIRECTORY = './../atom_files' # directories for input atom files
# index of the first loop model:
SET LOOP_STARTING_MODEL = 20
# index of the last loop model:
SET LOOP_ENDING_MODEL = 23
SET LOOP_MD_LEVEL = 'refine_1'   # the loop refinement method (1 fast / 3 slow)

CALL ROUTINE = 'loop'

# This routine picks model residues that need to be refined (necessary):

SUBROUTINE ROUTINE = 'select_loop_atoms'
  # Uncomment if you also want to optimize the loop environment:
  # SET SELECTION_SEARCH = 'SPHERE_SEGMENT', SPHERE_RADIUS = 6

  # 4 residue insertion (1st loop):
  PICK_ATOMS SELECTION_SEGMENT = '19:' '28:', SELECTION_STATUS = 'initialize'

```

```

# 2 residue insertion (2nd loop):
# PICK_ATOMS SELECTION_SEGMENT = '46:' '55:', SELECTION_STATUS = 'add'

RETURN
END_SUBROUTINE

```

```

# This routine adds any special restraints (optional):
#
# SUBROUTINE ROUTINE = 'special_restraints'
# MAKE_RESTRAINTS RESTRAINT_TYPE = 'ALPHA', RESIDUE_IDS = '46:' '55:'
# RETURN
# END_SUBROUTINE

```

4. **I want to build a model of a chimeric protein based on two known structures. Alternatively, I want to build a multi-domain protein model using templates corresponding only to the individual domains.**

This can be accomplished using the standard modeling routine. The alignment should be as follows when the chimera is a combination of proteins A and B:

```

proteinA  aaaaaaaaaaaaaaaaaaaaaaaaaa-----
proteinB  -----bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
chimera   aaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

```

In the PIR format the alignment file is:

```

>P1;proteinA
structureX:proteinA
aaaaaaaaaaaaaaaaaaaaaaaaaaaa-----*
>P1;proteinB
structureX:proteinB
-----bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb*
>P1;chimera
sequence:chimera
aaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb*

```

If no additional information is available about the relative orientation of the two domains the resulting model will probably have an incorrect relative orientation of the two domains when the overlap between A and B is non-existing or short. To obtain satisfactory relative orientation of modeled domains in such cases, orient the two template structures appropriately before the modeling.

5. **I don't want to use one region of a template for construction of my model.**

The easiest way to achieve this is to not align that region of the template with the target sequence. If region 'bbbbbbb' of the template should not be used as a template for region 'eeee' of the target sequence the alignment should be like this:

```

template  aaaaaaaaaaaaaaaaaaaaaa----bbbbbbcccccccccccccccccccccccccc
target    ddddddddddddddddeeeee-----ffffffffffffffffffffffffffff

```

The effect of this alignment is that no homology-derived restraints will be produced for region 'eeee'.

6. **I want to define (additional) disulfide bonds in the target sequence because no equivalent disulfide bonds exist in any of the templates (in which case PATCH_SS_TEMPLATES cannot define them automatically).**

MODELLER can restrain disulfides in two ways: automatically (**PATCH_SS_TEMPLATES** or **PATCH_SS_MODEL**) and manually (**PATCH**).

If there is an equivalent disulfide bridge in any of the templates aligned with the target, the **PATCH_SS_TEMPLATES** command will generate appropriate disulfide bond restraints without any other input from the user. This command is run automatically by the 'model' script used for comparative modeling. The restraints include bond, angle and dihedral angle restraints. The SG — SG atom pair also becomes an excluded atom pair that is not checked for an atom–atom overlap. The χ_i dihedral angle restraints will depend on the conformation of the equivalent disulfides in the template structure, as described in [?]. The command **PATCH_SS_MODEL** is similar, except that the current structure of MODEL, not templates, is used to guess the disulfide bonded CYS – CYS pairs.

Explicit manual restraints can be added by the **PATCH** command relying on the PRES DISU patching residue in the CHARMM topology file. This command is used by the 'special_patches' routine that is called automatically by the 'model' script. In comparative modeling by 'model', the 'manual' disulfides should be defined in the 'special_patches' routine. The **PATCH** command will establish the correct stereochemistry by relying on the CHARMM topology file and parameters to restrain the disulfide bond.

It is better to use **PATCH_SS_TEMPLATES** than **PATCH** where possible because the dihedral angles are restrained more precisely by using the templates than the general rules of stereochemistry.

Some CHARMM parameter files have a multiple dihedral entry for the disulfide dihedral angle χ_3 that consists of three individual entries with periodicities of 1, 2 and 3. This is why you see three feature restraints for a single disulfide in the output of the **ENERGY** command.

```
# This is as usual:
INCLUDE
SET ALIGNMENT_FILE = 'align1.ali', KNOWN='templ1', SEQUENCE='targ1'
CALL ROUTINE = 'model'
STOP
# Redefine the special_patches routine to include the additional disulfides
# (this routine is empty by default):
SUBROUTINE ROUTINE = 'special_patches'
  # A disulfide between residues 1 and 85 in chain A:
  PATCH RESIDUE_TYPE = 'DISU', RESIDUE_IDS = '1:A' '85:A'
  # A disulfide between residues 41 and 45 in chain B:
  PATCH RESIDUE_TYPE = 'DISU', RESIDUE_IDS = '41:B' '45:B'
  RETURN
END_SUBROUTINE
```

7. I want to explicitly force certain Pro residues to the *cis* ω conformation.

MODELLER should usually be allowed to handle this automatically *via* the omega dihedral angle restraints, which are calculated by default.

```
# This is as usual:
INCLUDE
SET ALIGNMENT_FILE = 'align1.ali', KNOWN='templ1', SEQUENCE='targ1'
CALL ROUTINE = 'model'
STOP
# Redefine the special_patches routine to force Pro to cis conformation:
# (this routine is empty by default):
SUBROUTINE ROUTINE = 'special_restraints'
  CALL ROUTINE = 'cispeptide', ATOM_IDS1 = 'O:4' 'C:4' 'N:5' 'CA:5', ;
  ATOM_IDS2 = 'CA:4' 'C:4' 'N:5' 'CA:5'
  RETURN
END_SUBROUTINE
```

8. How can I select/remove/add a set of restraints?

Restraints can be read from a file by **READ_RESTRAINTS**, calculated by **MAKE_RESTRAINTS**, or added “manually” by **ADD_RESTRAINT**. **PICK_RESTRAINTS** picks those restraints for objective function calculation that restrain the selected atoms only, as specified in the selected atoms set 1. Initially, all atoms are selected; this can be changed by the **PICK_ATOMS** command. **MAKE_RESTRAINTS** command for some restraint types (*e.g.*, distance) constructs restraints of the selected type between the atoms in the selected atoms sets 2 and 3. Script `'scripts/_homcsr.top'` contains examples of the **PICK_ATOMS** command when generating restraints by **MAKE_RESTRAINTS**. There are also commands for adding and deleting single restraints, **ADD_RESTRAINT** and **DELETE_RESTRAINT**, respectively. If you do **CONDENSE_RESTRAINTS**, the unselected restraints will be deleted. This is useful for getting rid of the unwanted restraints completely.

9. I want to add my own restraints for optimization of the model.

You can read your restraints whenever the default restraints are read.

```
INCLUDE
SET ALIGNMENT_FILE = 'align1.ali', KNOWNS='templ1', SEQUENCE='targ1'
CALL ROUTINE = 'model'
STOP
# Redefine the rd_restraints routine:
SUBROUTINE ROUTINE = 'rd_restraints'
  # This is the default homology-derived restraints:
  READ_RESTRAINTS FILE = CSRFILE, ADD_RESTRAINTS = off
  # This is two additional user provided files:
  READ_RESTRAINTS FILE = 'my_rsrs1.rsr', ADD_RESTRAINTS = on
  READ_RESTRAINTS FILE = 'my_rsrs2.rsr', ADD_RESTRAINTS = on
  SET ADD_RESTRAINTS = off
  RETURN
END_SUBROUTINE
```

10. I want to add my own restraints to the file with the automatically derived homology restraints, immediately after the default calculation of the homology-derived restraints.

This is achieved by redefining the `'special_restraints'` routine, which is empty by default.

```
INCLUDE
SET ALIGNMENT_FILE = 'align1.ali', KNOWNS='templ1', SEQUENCE='targ1'
CALL ROUTINE = 'model'
# Redefine the special_restraints routine:
SUBROUTINE ROUTINE = 'special_restraints'
  # Add some restraints from a file to existing homology-derived restraints:
  READ_RESTRAINTS FILE = 'my_rsrs1.rsr', ADD_RESTRAINTS = on
  # Restrain the specified CA-CA distance to 10 angstroms (st.dev.=0.1).
  # Use a harmonic potential and X-Y distance group.
  SET ATOM_IDS 'CA:35:A' 'CA:40:A'
  ADD_RESTRAINT RESTRAINT_PARAMETERS = 3 1 1 27 2 2 0 10.0 0.1
  SET ADD_RESTRAINTS = off
  RETURN
END_SUBROUTINE
```

11. I have my own restraints file to be used exclusively for optimization by the default comparative modeling routine.

```
INCLUDE
SET ALIGNMENT_FILE = 'align1.ali', KNOWNS='templ1', SEQUENCE='targ1'
SET CSRFILE = 'targ1.rsr', CREATE_RESTRAINTS = 0
CALL ROUTINE = 'model'
```

12. **I have my own initial structure to be used for optimization by the default comparative modeling routine.**

```
INCLUDE
SET ALIGNMENT_FILE = 'align1.ali', KNOWNNS='templ1', SEQUENCE='targ1'
# Specify the initial structure filename, and tell the program to
read the initial file, not construct it from the templates:
SET MODEL = 'targ1.ini', GENERATE_METHOD = 'read_xyz'
CALL ROUTINE = 'model'
```

13. **What are the different refinement levels really doing?**

There are two different optimization approaches available within MODELLER: variable target function method (VTFM) with conjugate gradients (CG) [?] and molecular dynamics (MD) with simulated annealing (SA) [?]. They can both be done to a different degree (with more or less cycles of CG and MD, faster or slower schedule for VTFM and SA). The exact details are best obtained from the scripts themselves because a detailed description would probably be longer than the scripts. For example, the QUANTA and INSIGHTII implementations of MODELLER have these three levels of optimization: no optimization (only copying coordinates from templates and building the undefined atoms using internal geometry information from the RTF entries); only VTFM with CG; also MD with SA. Most of the time (70%) is spent on the MD&SA part. Our experience is that when MD&SA are used, if there are violations in the best of the 10 models, they probably come from an alignment error, not an optimizer failure (if there are no insertions longer than approximately 15 residues).

14. **I want to change the default optimization schedule.**

See file 'scripts/_defs.top' for the variables that could be changed and for their possible values.

```
INCLUDE
SET ALIGNMENT_FILE = 'align1.ali', KNOWNNS='templ1', SEQUENCE='targ1'
# Very thorough VTFM optimization:
SET LIBRARY_SCHEDULE = 1, MAX_VAR_ITERATIONS = 300
# Very thorough MD optimization:
SET MD_LEVEL = 'refine1'
# Repeat the whole cycle 3-times and do not stop unless obj.func. > 1E6
SET REPEAT_OPTIMIZATION = 3, MAX_MOLPDF = 1E6
CALL ROUTINE = 'model'
```

15. **I want to build an all hydrogen atom model with water molecules and other non-protein atoms (atoms in the HETATM records in the PDB file).**

```
INCLUDE
SET ALIGNMENT_FILE = 'align1.ali', KNOWNNS='templ1', SEQUENCE='targ1'
SET MODEL_TOPOLOGY = 1, HYDROGEN_IO = on, HETATM_IO = on, WATER_IO = on
SET TOPOLOGY_LIBRARY = $(LIB)/top.lib
SET PARAMETERS_LIBRARY = $(LIB)/par.lib
CALL ROUTINE = 'model'
```

16. **How do I build a model with water molecules or residues that do not have an entry in the topology and/or parameter files?**

Water molecules are indicated by 'w' in the alignment file and the special block residue ('BLK') that does not have entries in the residue topology and parameter libraries is indicated by '.'.

See Section 2.2.1 for information about block residues.

```
INCLUDE
SET ALIGNMENT_FILE = 'align1.ali', KNOWNNS='templ1', SEQUENCE='targ1'
SET HETATM_IO = on, WATER_IO = on
CALL ROUTINE = 'model'
```

The alignment file:

```
>P1;templ1
structureX:templ1:1::10::
FAYVI/.wwww*
```

```
>P1;targ1
sequence:targ1:1::8::
-GWIV/.ww-w*
```

17. How do I define my own residue types, such as D-amino acids, special ligands, and unnatural amino-acids?

This is a painful area in all molecular modeling programs. However, CHARMM and X-PLOR provide a reasonably straightforward solution *via* the residue topology and parameter libraries. MODELLER uses CHARMM topology and parameter library format and also extends the options by allowing for a generic “BLK” residue type (Section 2.2.1). This BLK residue type circumvents the need for editing any library files, but it is not always possible to use it. Due to its conformational rigidity, it is also not as accurate as a normal residue type. In order to define a new residue type in the MODELLER libraries, you have to follow the series of steps described below. As an example, we will define the ALA residue without any hydrogen atoms. You can add an entry to the MODELLER topology or parameter file; you can also use your own topology or parameter files. For more information, please see the CHARMM manual.

- (a) Define the new residue entry in the residue topology file (RTF), say 'top_heav.lib'.

```
RESI ALA      0.00000
ATOM N      NH1    -0.29792
ATOM CA     CT1     0.09563
ATOM CB     CT3    -0.17115
ATOM C      C       0.69672
ATOM O      O      -0.32328
BOND CB CA   N CA   O C   C CA   C +N
IMPR C CA +N O   CA N C CB
IC -C   N   CA   C           1.3551 126.4900 180.0000 114.4400 1.5390
IC N    CA   C   +N         1.4592 114.4400 180.0000 116.8400 1.3558
IC +N   CA  *C   O          1.3558 116.8400 180.0000 122.5200 1.2297
IC CA   C   +N  +CA        1.5390 116.8400 180.0000 126.7700 1.4613
IC N    C   *CA  CB        1.4592 114.4400 123.2300 111.0900 1.5461
IC N    CA   C   O          1.4300 107.0000   0.0000 122.5200 1.2297
PATC FIRS NTER LAST CTER
```

You can obtain an initial approximation to this entry by defining the new residue type using the residue type editor in QUANTA and then writing it to a file.

The RESI record specifies the CHARMM residue name, which can be up to four characters long and is usually the same as the PDB residue name (exceptions are the potentially charged residues where the different charge states correspond to different CHARMM residue types). The number gives the total residue charge.

The ATOM records specify the IUPAC (*i.e.*, PDB) atom names and the CHARMM atom types for all the atoms in the residue.¹ The number at the end of each ATOM record gives the partial atomic charge.

The BOND records specify all the covalent bonds between the atoms in the residue (*e.g.*, there are bonds CB-CA, N-CA, O-C, etc.). In addition, symbol '+' is used to indicate the bonds to the subsequent residue in the chain (*e.g.*, C - +N). The covalent angles and dihedral angles are calculated automatically from the list of chemical bonds.

¹There are several small differences between the “CHARMM” IUPAC definitions here and those actually used by PDB (*e.g.*, Ile CD1 in PDB is CD in CHARMM, Leu CD1 and CD2 atoms are swapped, and the PDB carboxy terminal atoms O and OXT are OT1 and OT2 in CHARMM, respectively). These differences are handled internally by MODELLER.

The IMPR records specify the improper dihedral angles, generally used to restrain the planarity of various groups (*e.g.*, peptide bonds and sidechain rings). See also below.

The IC (internal coordinate) records are used for constructing the initial Cartesian coordinates of a residue. An entry

$$IC \quad a \quad b \quad c \quad d \quad d_{ab} \quad \alpha_{abc} \quad \Theta_{abcd} \quad \alpha_{bcd} \quad d_{cd}$$

specifies distances d , angles α , and either dihedral angles or improper dihedral angles Θ between atoms a , b , c and d , given by their IUPAC names. The improper dihedral angle is specified when the third atom, c , is preceded by a star, '*'. As before, the '-' and '+' pre-fixes for the atom names select the corresponding atom from the preceding and subsequent residues, respectively. The distances are in angstroms, angles in degrees. The distinction between the dihedral angles and improper dihedral angles is unfortunate since they are the same mathematically, except that by convention when using the equations, the order of the atoms for a dihedral angle is $abcd$ and for an improper dihedral angle it is $acbd$.

The PATC record specifies the default patching residue type when the current residue type is the first or the last residue in a chain.

- (b) You have to make sure that all the CHARMM atom types of the new residue type occur in the MASS records at the beginning of the topology library: Add your entry at the end of the MASS list if necessary. If you added any new CHARMM atom types, you also have to add them to the radii libraries, 'modlib/radii.lib' and 'modlib/radii14.lib'. These libraries list the atomic radii for the different topology models, for the long range and 1-4 non-bonded soft-sphere terms, respectively. The full names of the files that are used during calculation are given by the environment variables \$RADII_LIB and \$RADII14_LIB.
- (c) Optionally, you can add the residue entry to the library of MODELLER topology models, 'modlib/models.lib'. The runtime version of this library is specified by the environment variable \$MODELS_LIB. This library specifies which subsets of atoms in the residue are used for each of the possible topologies. Currently, there are 9 topologies selected by MODEL_TOPOLOGY (3 is default):
- | | | |
|---|------|--|
| 1 | ALLH | all atoms |
| 2 | POL | polar hydrogens only |
| 3 | HEAV | non-hydrogen atoms only |
| 4 | MCCB | non-hydrogen mainchain (N, C, CA, O) and CB atoms |
| 5 | MNCH | non-hydrogen mainchain atoms only |
| 6 | MCWO | non-hydrogen mainchain atoms without carbonyl O |
| 7 | CA | CA atoms only |
| 8 | MNSS | non-hydrogen mainchain atoms and disulfide bonds |
| 9 | CA3H | reduced model with a small number of sidechain interaction centers |

The Ala entry is:

```
#
      ALLH POLH HEAV MCCB MNCH MCWO CA  MNSS CA3H
*
RESI ALA
ATOM  NH1  NH1  NH1  NH1  NH1  NH1  ##### NH1  #####
ATOM  H    HN   ##### ##### ##### ##### ##### #####
ATOM  CT1  CT1  CT1  CT1  CT1  CT1  CT1  CT1  CAH
ATOM  HB   ##### ##### ##### ##### ##### ##### ##### CH3E
ATOM  CT3  CT3  CT3  CT3  ##### ##### ##### ##### #####
ATOM  HA   ##### ##### ##### ##### ##### ##### ##### #####
ATOM  HA   ##### ##### ##### ##### ##### ##### ##### #####
ATOM  HA   ##### ##### ##### ##### ##### ##### ##### #####
ATOM  C    C    C    C    C    C    ##### C    #####
ATOM  O    O    O    O    O    ##### ##### O    #####
```

The residue entries in this library are separated by stars. The '####' string indicates a missing atom. The atom names for the present atoms are arbitrary. The order of the atoms must be the same as in

the CHARMM residue topology library. If a residue type does not have an entry in this library, all atoms are used for all topologies.

- (d) You have to add the new residue type to the residue type library, 'modlib/restyp.lib'. The execution version of this file is specified by the environment variable \$RESTYP_LIB. For the ALA residue,

```
1 | ALA          | A | ALA | alanine
```

You would generally add the new residue type at the end of the file. There are 5 fields in each line, separated by the '|' characters. The first field is an integer index corresponding to the integer residue type. The standard residue types have their indices smaller than 24. These are also the indices corresponding to the residue-residue substitution matrices. The second field contains the list of equivalent PDB or IUPAC 3-character residue names, used in the PDB files. A list rather than a single name is allowed because PDB can unfortunately use different names for the same residue type (*e.g.*, water can be HOH, WAT, etc.). The third field gives a single character code for the residue type, which is used in the alignment file. This does not have to be unique, but if it is not unique you cannot use it in the alignment file. Any ASCII character is fine, it does not have to be a letter. If you run out of characters you can re-define the existing ones that you do not need. The fourth field gives the four-character CHARMM residue name, as specified in the RESI record of the topology library. The last field contains an optional comment.

Every residue in the CHARMM topology file has to have an entry in the \$RESTYP_LIB library, but not every residue entry in the \$RESTYP_LIB library needs an entry in the residue topology file.

When you are adding a new residue type, you have to hope that the maximal number of residue types is not over-reached. If it is, a fatal error is reported at the beginning of the execution. To solve this problem, you could delete some of the un-needed existing residue types in the \$RESTYP_LIB file, rather than re-compile the program with larger array sizes. You can also read your own residue type library by the **READ_RESTYP_LIB** command.

- (e) In general, when you add a new residue type, you also add new chemical bonds, angles, dihedral angles, improper dihedral angles, and non-bonded interactions, new in the sense that a unique combination of CHARMM atom types is involved whose interaction parameters are not yet specified in the parameter library (see also Section 2.2.1). In such a case, you will get a number of warning and/or error messages when you generate the stereochemical restraints by the **MAKE_RESTRAINTS** command. These messages can sometimes be ignored because MODELLER will guess the values for the missing parameters from the current Cartesian coordinates of the model. When this is not accurate enough or if the necessary coordinates are undefined you have to specify the parameters explicitly in the parameter library. Search for BOND, ANGL, DIHE, and IMPR sections in the parameters library file and use the existing entries to guess your new entries. Note that you can use dummy atom types 'X' to create general dihedral (*i.e.*, X A A X) and improper dihedral angle (*i.e.*, A X X A) entries, where A stands for any of the real CHARMM atom types. For the dihedral angle cosine terms, the CHARMM convention for the phase is different for 180° from MODELLER's (Eq. 5.56). If you use non-bonded Lennard-Jones terms, you also have to add a NONB entry for each new atom type. If you use the default soft-sphere non-bonded restraints, you have already taken care of it by adding the new atom types to the \$RADII_LIB and \$RADII_LIB libraries.

18. How do I define my own patching residue types?

This is even messier than defining a new residue type. As an example, we will define the patching residue for establishing a disulfide bond between two CYS residues.

```
PRES DISU          -0.36 ! Patch for disulfides. Patch must be 1-CYS and 2-CYS.
ATOM 1CB CT2      -0.10 !
ATOM 1SG SM       -0.08 !           2SG--2CB--
ATOM 2SG SM       -0.08 !           /
ATOM 2CB CT2     -0.10 ! -1CB--1SG
DELETE ATOM 1HG1
DELETE ATOM 2HG1
BOND 1SG 2SG
IC 1CA 1CB 1SG 2SG 0.0000 0.0000 180.0000 0.0000 0.0000
```



```

IC 1CB 1SG 2SG 2CB 0.0000 0.0000 90.0000 0.0000 0.0000
IC 1SG 2SG 2CB 2CA 0.0000 0.0000 180.0000 0.0000 0.0000

```

The PRES record specifies the CHARMM patching residue type (up to four characters). As for the normal RESI residue types, patching residue types also have to be defined in the residue type library, 'modlib/restyp.lib'.

The ATOM records have the same meaning as for the RESI residue types described above. The extension is that the IUPAC atom names (listed first) must be pre-fixed by the index of the residue that is patched. In this example, there are two CYS residues that are patched, thus the prefixes 1 and 2. When using the **PATCH** command, the order of the patched residues specified by RESIDUE_IDS must correspond to these indices (this is only important when the patch is not symmetric, unlike the 'DISU' patch in this example).

DELETE records specify the atoms to be deleted, the two hydrogens bonded to the two sulphurs in this case.

The BOND and IC (internal coordinate) records are the same as those for the RESI residues, except that the atom names are prefixed with the patched residue indices.

19. Is it possible to restrain secondary structure in the target sequence?

Yes. There are 'ALPHA', 'STRAND' and 'SHEET' restraint types that the **MAKE_RESTRAINTS** command can generate. One specifies the segment which is then restrained to the specified secondary structure conformation. For example,

```

# This is as usual:
INCLUDE
SET ALIGNMENT_FILE = 'align1.ali', KNOWN='templ1', SEQUENCE='targ1'
CALL ROUTINE = 'model'
STOP
# Redefine the special_restraints routine to include the secondary
# structure restraints (this routine is empty by default):
SUBROUTINE ROUTINE = 'special_restraints'
  SET ADD_RESTRAINTS = on
  # An alpha-helix:
  MAKE_RESTRAINTS RESTRAINT_TYPE = 'alpha', RESIDUE_IDS = '20' '30'
  # SET KEEP_DUPL_RESTR = 'new'
  # Two strands:
  MAKE_RESTRAINTS RESTRAINT_TYPE = 'STRAND', RESIDUE_IDS = '1' '6'
  MAKE_RESTRAINTS RESTRAINT_TYPE = 'STRAND', RESIDUE_IDS = '9' '14'
  # An anti-parallel sheet:
  MAKE_RESTRAINTS RESTRAINT_TYPE = 'SHEET', ATOM_IDS = 'N:1' 'O:14', SHEET_H-BONDS = -5
  RETURN
END_SUBROUTINE

```

20. I want to patch the N-terminal or (C-terminal) residue (e.g., to model acetylation properly), but the PATCH command does not work.

This is probably because the N-terminus is patched by default with the NTER patching residue (corresponding to -NH₃⁺) and a patched residue must not be patched again. The solution is to turn the default patching off by SET PATCH_DEFAULT = off before the **GENERATE_TOPOLOGY** command is called.

21. Is it possible to use templates with the coordinates for C_α atoms only?

Yes. You do not have to do anything special.

22. How do I analyze the output log file?

First, check for the error messages by searching for string '_E>'. These messages can only rarely be ignored. Next, check for the warning messages by searching for string '_W>'. These messages can almost always be ignored. If everything is OK so far, the most important part of the log file is the output of the **ENERGY** command for each model. This is where the violations of restraints are listed. When there are too many too violated restraints, more optimization or a different alignment is needed. What is too many and too much?

It depends on the restraint type and is best learned by doing **ENERGY** on an X-ray structure or a good model to get a feel for it. You may also want to look at the output of command **CHECK_ALIGNMENT**, which should be self-explanatory. I usually ignore the other parts of the log file.

23. How do I prevent “knots” in the final models?

The best way to prevent knots is to start with a starting structure that is as close to the desired final model as possible. Other than that, the only solution at this point is to calculate independently many models and hope that in some runs there won't be knots. Knots usually occur when one or more neighboring long insertions (*i.e.*, longer than 15 residues) are modeled from scratch. The reason is that an insertion is build from a randomized distorted structure that is located approximately between the two anchoring regions. Under such conditions, it is easy for the optimizer to “fall” into a knot and then not be able to recover from it. Sometimes knots result from an incorrect alignment, especially when more than one template is used. When the alignment is correct, knots are a result of optimization not being good enough. However, making optimization more thorough by increasing the CPU time would not be worth it on the average as knots occur relatively infrequently. The excluded volume restraints are already included in the standard comparative modeling routine.

24. What do I do when I get Syntax error at line 1: ‘(’ unexpected message?

The executable is not recognized as such on your system. Make sure you FTP the file in the binary format. Make sure the system version matches the self-descriptive name of the binary file. Also it could be related to automatic processing of files by some Web browsers. Make sure you got a binary, not the file compressed by “compress” or “gzip” command. If you are not sure about the version of your system use the most generic executable which has been compiled for lower version of operating system.

25. What is considered to be the minimum length of a sequence motif necessary to derive meaningful constraints from the alignment to use in modeling.. one, two, three, or more?

Usually more than that (dozens if you want just to detect reliable similarity, and even more if you want a real model). It is good to have at least 35-40 Sometimes even 30

1.10 MODELLER updates

1.10.1 Major changes in Releases 5 and 6

The last generally released academic version of MODELLER was version 4. MODELLER 5 has not been generally released. The current version is MODELLER 6.

The major changes relative to version 4 include (this is a very incomplete list):

- New TOP loop modeling routine 'loop' significantly improves the accuracy of loop modeling (Section 3.3). It can also be used in modeling of sidechains or other parts of the structure.
- Several new statistical atomic distance-dependent potentials can typically be used for loop modeling and model evaluation.
- The ENERGY OUTPUT = 'ENERGY_PROFILE' allows construction of energy profiles for model evaluation, based on any combination of MODELLER energy terms. It replaces the **ENERGY_PROFILE** command.
- New functional form and parameters for the binormal Φ, Ψ restraints improve their numerical stability and accuracy.
- New TOP command **SEGMENT_MATCHING** facilitates exploring many different alignments in the difficult comparative modeling cases.
- New dynamic memory allocation makes MODELLER more memory efficient.
- Many bug fixes.
- Many new arguments, changed naming/meaning of arguments, and several new commands.
- Conversion from Fortran 77 to Fortran 90 was the main culprit for a very long delay before the latest release.

1.10.2 Detailed chronological listing of some changes since MODELLER-4 release on 17 June, 1997

For an incomplete list of the changes from 17 March 1994 to the release of MODELLER-4, see file 'doc/modeller4-changes'.

- 6/23/97, AŠ: Added ALIGNMENT_FORMAT = 'FASTA' format to the **WRITE_ALIGNMENT** command (routines rda1n and rdp1r).
- 6/24/97, AŠ: Corrected the bug in **OPERATE** which sometimes caused errors in using the result variable (*e.g.*, with **WRITE OBJECTS = RESULT**) that has been defined by the user and not assigned a scalar value before the call to **OPERATE** (routine act13).
- 6/25/97, AŠ: Incorporated several minor code changes suggested by Andrzej Sawaryn to allow compilation with f90 on Ultra 2 running Solaris 2.5.1 (routine ran6, file 'Makefile').
- 6/27/97, AŠ: Changed **BUILD BUILD_METHOD = '3D_INTERPOLATION'** so that the new "line" of atoms is a little randomized to prevent numerical problems with energy calculations (linear impropers, angles, *etc.*) (routine insxyz).
- 6/27/97, AŠ: Upgraded **TRANSFER_XYZ** by adding CLUSTER_METHOD, which selects how are the template–template comparisons done at each residue position ('RMSD' or 'MAXIMAL_DISTANCE') (routine trfxyz).
- 7/11/97, AŠ: Changed the format of the representation of spline restraints in the .rsr files (MODELLER12 format changes to MODELLER5 format). The specs now contain only the *y* values, the derivatives are calculated on the fly. The same for the non-bonded MODELLER restraints in the parameter file.

- 7/13/97, AŠ: Added arguments `SIGNIF_CUTOFF` and `SEARCH_GROUP_LIST` to the `SEQUENCE_SEARCH` command. All members of the groups that have `SIGNIF` larger than `SIGNIF_CUTOFF[1]` and are not more than `SIGNIF_CUTOFF[2]` units worse than the best hit are put into the alignment arrays.
- 7/13/97, AŠ: Added structure resolution to the code in the output pairwise distance matrix produced by the `ID_TABLE` and `SEQUENCE_COMPARISON` commands.
- 7/19/97, AŠ: Added a check for sufficient overlap in `MALIGN3D`; not a stop, but a return in that case now.
- 7/22/97, AŠ: Added the `READ_RESTYP_LIB` command for reading new residue type definitions, and the corresponding `RESTYP_LIB.FILE` variable for specifying the library name.
- 7/22/97, AŠ: Simplification of the 'mod5' script.
- 7/22/97, AŠ: Added automatic recognition of the R, E, and P entries to the `ADD_RESTRAINT` command (it was impossible to add the excluded pairs and pseudo atom definitions before this modification).
- 7/30/97, AŠ: Added keyword `SWAP_ATOMS_IN_RES` to `SUPERPOSE`. This allows swapping atoms in residues 'DEFHLNQRVY' (as selected) to minimize sidechain RMS deviation between `MODEL` and `MODEL2` that have exactly the same atoms in the same order (`match.F`).
- 8/8/97, AŠ: Changed the `_model.top` script so that the `FINAL_MALIGN3D` option works for alignments that use the `.` option for segment specification.
- 8/9/97, AŠ: Renamed the `KEEP_RESTRAINTS` string variable to the `RESTRAINT_SEL_ATOMS` integer variable and generalized its meaning (see below). Old `KEEP_RESTRAINTS` of `ONE_ATOM` (default) is equal to the new `RESTRAINT_SEL_ATOMS` of 1; and the old `KEEP_RESTRAINTS` of `ALL_ATOMS` is equal to the new `RESTRAINT_SEL_ATOMS` of 9999. Impacts `MAKE_RESTRAINTS` and `PICK_RESTRAINTS`.
- 8/9/97, AŠ: Renamed the `DYNAMIC_FLAG` variable to `NONBONDED_SEL_ATOMS` and modified its meaning in order to make it consistent with the new variable `RESTRAINT_SEL_ATOMS`. Old `DYNAMIC_FLAG` of 0 (default) is equal to the new `NONBONDED_SEL_ATOMS` of 2; and the old `DYNAMIC_FLAG` of 1 is equal to the new `NONBONDED_SEL_ATOMS` of 1. Impacts `MAKE_RESTRAINTS` and `OPTIMIZE`.
- 8/9/97, AŠ: Added `RESTRAINT_SEL_ATOMS` and modified the `MAKE_RESTRAINTS` command so that the stereochemical, MRFP stereochemical, dihedral angle, and bi-dihedral angle restraints are calculated only when a sufficiently large number of atoms in a possible restraint are selected (set 1): For a given possible restraint, only when all restraint atoms or at least `RESTRAINT_SEL_ATOMS` are selected (set 1), is the restraint calculated. An exactly equivalent role for the 'static' non-bonded restraints (and also for the 'dynamic' non-bonded restraints calculated during `OPTIMIZE`) is played by `NONBONDED_SEL_ATOMS` (the new `TOP` variable that replaces and modifies the meaning of the old, deleted `TOP` variable `DYNAMIC_FLAG`). This now makes it possible to restrict the calculation of all new restraints to the selected atoms only (atom-atom distance restraints already depended on selected atom sets 2 and 3; secondary structure restraints are calculated for explicitly defined chain segments only).
- 8/9/97, AŠ: Modified the `MAKE_RESTRAINTS` command so that the calculation of the 'static' non-bonded restraints now takes into account `RESIDUE_SPAN_RANGE` and `RESIDUE_SPAN_SIGN`.
- 8/9/97, AŠ: Corrected a bug in `MAKE_RESTRAINTS` (`'mkdyn.F'`) that resulted in an incorrect use of `DYNAMIC_FLAG` during calculation of the 'static' non-bonded restraints.
- 8/9/97, AŠ: Optimized speed of `MAKE_RESTRAINTS` for non-bonded restraints by considering the selected atoms (set 1) instead of all atoms.
- 8/9/97, AŠ: Optimized speed of `PICK_RESTRAINTS` by using the `'picatm'` array instead of the `'cnsin1'` and `'cnsin2'` functions.
- 8/9/97, AŠ: Corrected a bug in `'preppdf.F'` that sometimes resulted in a loss of explicitly excluded non-bonded pairs when these were not defined or read in before each call to `'preppdf.F'`.

- 9/7/97, AŠ: Initialized the 'ialn()' array for the PAP, INSIGHT, and QUANTA formats ('rdaln.F'). This bug may have caused some problems when a short protein is added to the alignment that contained a longer protein (framework) in its place (*e.g.*, with **READ_ALIGNMENT** after **MALIGN3D**).
- 9/13/97, AŠ: Renamed TOP variable MODELLER_STATUS into ERROR_STATUS and moved it within the top.ini file.
- 9/16/97, AŠ: Modified protsize.cst and Makefile to allow more flexibility in defining the sizes of the most important arrays.
- 9/20/97, AŠ: Introduced the ALIGN_BLOCK argument for the **MALIGN** command.
- 9/28/97, AB: New parameters:ASGL_2D_TYPE – controls the way 2D arrays are output by the **ENERGY_PROFILE** command. PROFILE_2D_PHYS – selects physical type to be presented as a 2D energy profile. NORMALIZE – selects normalization for **ENERGY_PROFILE**. ASGL_2D_OUTPUT – whether or not to write 2D output for ASGL. Command names changed: **PATCH_DISULFIDES** to **PATCH_SS_MODEL**. **INHERIT_DISULFIDES** – to **PATCH_SS_TEMPLATES**. **ENERGY_PROFILE** now outputs 2D profiles. OUTPUT can contain the type of energy you output by **ENERGY_PROFILE**: ENERGY or IDEAL_ENERGY1, and NORMALIZE specifies the type of normalization. Removed the error in calculation of the spline non-bonded terms.
- 10/28/97, AB: the bug in rsr9 (tmps size) fixed. Restored output of the objective function to the output PDB file.
- 1/2/98, AB: version 5f-ab: Introduced 32-nd physical restraint type and 7-th feature type (accessible surface in Å²). Libraries 'sched.lib', 'feats.lib', and 'top.ini' are changed to reflect this. New logical variable DYNAMIC_ACCESS was introduced. Limitation, each time before you want to use the 32-nd feature in **ENERGY** etc., you have to calculate solvent accessibility by **WRITE_DATA OUTPUT=PSA**.
- 1/5/98, AŠ: A bug (causing a crash) in seqsearch.F is fixed.
- 1/20/98, AB: version 5g = merge (5f-ab, 5f-as). To use solvent accessibility: **READ_PARAMETERS FILE = \$LIB/par-32.lib**
- 2/6/98, AB: version 5g/doc: Improved the html version of the manual. Changes in Makefile. latex2html was patched to properly handle the macros containing sectioning commands.
- 1/5/98, AŠ: Added keyword DIH_LIB_ONLY to **MAKE_RESTRAINTS**, to allow forcing the calculation of the “homology-derived” mainchain and sidechain dihedral angle restraints (single and binormal) from the residue type dependent libraries only (templates ignored).
- 1/25/98, AŠ: changed default BUILD_METHOD back to INTERNAL_COORDINATES.
- 2/19/98, AŠ: corrected a bug in distdp3.inc which sometimes caused incorrect C-terminal overhang treatment, possibly incorrect alignment as a whole.
- 2/27/98, AŠ: MNCH2_LIB moved to MNCH_LIB.
- 3/05/98, AŠ: Changed the algorithm for calculating the mainchain curvature. Now using least-squares lines instead of simple C_α-C_α vectors.
- 3/12/98, AŠ: added option SIGNIF_CUTOFF to **SEQUENCE_SEARCH**.
- 3/23/98, AŠ: changed ALIGN_CODES to ALIGN_CODES2 for **READ_ALIGNMENT2**.
- found a workaround for a nasty compiler bug in mod2ali (for SGI, make opts) that caused naln=1.
- 4/28/98, AŠ: added MNCH?_LIB libraries to allow for i-1,i,i+1 mainchain conformation restraints.
- 4/30/98, AB: in 'model/build.F': all DISU stuff was deleted. atom classes are assigned based on the iss array, resnam does not change.
- 4/30/98, AB: In energy calculations, "slow" character parameters were replaced by "fast" integer ones.

- 6/6/98, AŠ: Created new command **SEGMENT_MATCHING** for enumerating the alignments.
- 6/6/98, AŠ: Removed **SELECTION_STEP** from **MODELLER**.
- 6/6/98, AŠ: Expanded options to **PICK_ATOMS** by allowing for automated selection of loops; a special value for **SELECTION_SEGMENT** of 'LOOPS' and new variable **GAP_EXTENSION** were introduced.
- 6/6/98, AŠ: Modified the loop modeling scripts and other scripts so that robust loop modeling can be done automatically.
There are at least 43 such arrays with a dozen different character lengths.
- 6/6/98, AŠ: Added the single mutant modeling script.
- 7/1/98, AŠ: Added the missing **ALIGN_ALIGNMENT** top 'top.ini'.
- 7/1/98, AŠ: Removed undefined entries from **top.ini**. Changed the meaning of **RENUMBER_RESIDUES** so that it corresponds to the first residue index in renumbering.
- 7/1/98, AŠ: removed **sdch.lib** for sidechain classes 1–2 from all parts, including **COMPARE**.
- 7/1/98, AŠ: changed format of **mnch.lib**, removed class weights from this library because they occur in **mnch?.lib**.
- 7/7/98, AB: the multiple binormal restraints were debugged: 1) detection of most relevant phase of phi/psi angles 2) removal of the bug in calculation of the derivatives
- 7/8/98, AŠ: increased flexibility of **ALIGN2D** so that it needs only at least one structure for the first block of sequences.
- 7/9/98, AB: CA-only warning in **READ_MODEL** and other routines using **rdpdb**
- 7/11/98, AB: New kind of error message is introduced: **_B** - to inform about bugs. Now used for former "Increase MAX..." and similar messages only. **Scan_errors** modified respectively.
- 7/18/98, AŠ and AB: **BUILD_METHOD** for **BUILD_MODEL** can continue to be **3D_INTERPOLATION**, **INTERNAL_COORDINATE** but can now also assume **ONE_STICK** and **TWO_STICKS** values to generate different initial structures for inserted atoms.
- 7/20/98, AŠ: **biperiodic** is now used
- 7/24/98, AŠ: not using **schedule** in **PICK_RESTRAINTS** at all.
- 8/24/98, AŠ: introduced **EXPAND_CONTROL** to allow more control for the **EXPAND_ALIGNMENT** command.
- 8/24/98, AŠ: made a vector out of **RENUMBER_SEGMENTS**.
- 8/24/98, AŠ: introduced **NO_TER**.
- 9/10/98, AŠ: updated **COMPARE_SEQUENCES** so that variability of the first sequence in the alignment is written to the **Biso** column of **MODEL**.
- 9/15/98, AŠ: changed default **NONBONDED_SEL_ATOMS** from 2 to 1, in **top.ini**.
- 98–00, AŠ: too numerous changes to list here, conversion from **f77** to **f90**.

Chapter 2

MODELLER commands

Sections in this Chapter describe technical aspects of MODELLER. They include:

- miscellaneous rules and features of MODELLER (Section 2.1);
- dealing with stereochemical parameters and molecular topology (Section 2.2);
- handling of atomic coordinates (Section 2.3);
- comparing and searching of sequences and structures (Section 2.4);
- calculating spatial restraints (Section 2.5),
- deriving the model by minimizing the restraints (Section 2.6).

2.1 Miscellaneous rules and features of MODELLER

This Section describes several features of the program, including file naming conventions, various file types, and the control of the amount of output.

2.1.1 MODELLER system

One of the main aims of MODELLER is to allow for flexible exploration of various modeling protocols to facilitate the development of better modeling methods. MODELLER can be seen as an interpreted language that is specialized for modeling of protein 3D structure. MODELLER's organization is hierarchical and modular:

```
User script files.  
  MODELLER tasks (e.g., 'model').  
    Library of MODELLER's routines.  
      TOP interpreter commands.  
        TOP interpreter.  
          MODELLER source code.
```

2.1.2 Running MODELLER scripts

MODELLER is run by

```
mod script_file_name
```

where `script_file_name` is the name of the script file with instructions for MODELLER. This file contains commands in the TOP language. Each command line consist of the name of the command and optional variable assignments that control the action of the command. The scope of the variables is global; that is, once a variable is assigned on any command line, the assigned value remains in effect, in the main program and all subroutines, until explicitly changed by another assignment or, in a few cases, by MODELLER. All the commands and the default values of the variables are listed in Section 5.4. This Chapter describes the TOP commands that are used for dealing with proteins; the general TOP commands (*e.g.*, assignment, flow control, arithmetic operations) are described in Chapter 4.

See directory `examples` for examples of the TOP scripts that use commands described in this Chapter. In particular, sub-directory `examples/commands` contains the examples used in this Chapter. Another set of TOP scripts that you could use as templates can be found in the `scripts` directory.

2.1.3 Controlling breakpoints and the amount of output

Some errors are recoverable. For those errors, TOP variable `MODELLER_STATUS` becomes 1. A test is then performed: If `MODELLER_STATUS` is equal or greater then `STOP_ON_ERROR`, execution stops; otherwise, the control is passed back to the calling TOP routine where execution continues with the next TOP command. It is then up to your TOP script to deal sensibly with the failure of the preceding command. For example, this flexibility allows derivation of multiple models and searching for many sequences, even if some cases abort due to convergence problems.

There are five kinds of messages that MODELLER writes to the `log` file, indexed 1 to 5: long output from the MODELLER commands, short notes to do with the execution of the program (files opened, *etc.*), warnings identified by `'_W>'`, errors identified by `'_E>'`, and the messages about the status of dynamic memory allocation. The five elements in the TOP variable `OUTPUT_CONTROL[1:5]` can assume values of 0 or 'not 0'; 0 indicates that the corresponding information is not written out, 'not 0' indicates that it is.¹ Thus, different amounts of output can be selected. If everything is well, `OUTPUT_CONTROL = 1 0 0 1 0` is convenient because no execution messages, warnings, and dynamic memory reports are written out; for debugging, use `OUTPUT_CONTROL = 1 1 1 1 1`. To increase the detail of the dynamic memory status reports, set the last flag to 2.

2.1.4 File naming

There are several filename generating mechanisms that facilitate file handling. Not all of them apply to all file types.

Environment variables

There can be UNIX shell environment variables in any input or output filename. The environment variables have to be in the format `${VARNAME}` or `$(VARNAME)`. Also, four predefined macros are available for string variables:

- `${LIB}` is expanded into the `$LIB_MODELLER` variable defined in `modlib/libs.lib` (equal to `$MODINSTALL6/modlib`);
- `${DIR}` is expanded into the TOP variable `DIRECTORY`;
- `${JOB}` is expanded into the root of the TOP script filename;
- `${DEFAULT}` is expanded into `(ROOT_NAME) (FILE_ID) (ID1) (ID2) (FILE_EXT)`, where `ROOT_NAME`, `FILE_ID`, `ID1`, `ID2`, and `FILE_EXT` are TOP variables. `FILE_ID` is a string that may be set to `'default'`. In that case, a hard-wired short string is used instead of `FILE_ID`. Otherwise, the explicitly specified `FILE_ID` is applied. In any case, `FILE_ID` is not modified by the filename generation routine so that it can be used more than once without resetting it to the `'default'` value. Four digits are used for both `ID1` and `ID2`. For example, `'2ptn.B99990001'` results from `ROOT_NAME = '2ptn'`, `FILE_EXT = '.B'`, `ID1 = 9999`, and `ID2 = 1`.

¹This has not been implemented for all the output yet.

Automatic filename generation

For any filename, input or output, if the value of the variable is 'default' (case insensitive), the actual filename is constructed within the routine that will use the filename. The name is constructed by the same rule as that for the `DEFAULT` environment variable (Section 2.1.4). The only difference between the two cases is that `SET FILE = 'default'` may not work as expected if the `TOP` variables defining the filename change between the `SET` command and the command that will use the filename, whereas `SET FILE = '$DEFAULT'` will work as expected because the filename `FILE` is actually constructed during the `SET` command.²

Directory prefixes

Input. For many input filenames, the full filename is obtained by looking for the file in the list of directories specified in the `TOP` variable `DIRECTORY`. The directories in `DIRECTORY` are separated by colons (':') (e.g., 'dir1:dir2:dir3:...'). `DIRECTORY` can also contain the current directory ('' or './').

The directory prefix for the input atom coordinate filenames is obtained in a similar way, except that `ATOM_FILES_DIRECTORY` is used instead of `DIRECTORY`. Moreover, there is an additional mechanism for reading an atom coordinate file that requires specifying the protein code only (see below in Section on coordinate files and derivative data).

The list of directories is not scanned for the input filenames that start with '/'.

In contrast, the `INCLUDE_FILE` file is looked for in the distribution's `$BIN_MODELLER6` directory (equal to `$MODINSTALL6/bin` directory) in addition to the `DIRECTORY` directories. This allows for an easy inclusion of the predefined system `'_*.top'` files by the `INCLUDE` command.

Output. For all output filenames, except for those that start with '/', the full output filename is obtained by pre-fixing the filename with `OUTPUT_DIRECTORY`.

Coordinate files and derivative data

When accessing an atom file, a specified filename is tried first. If this is unsuccessful, MODELLER automatically expands the original filename by adding extension '.Z'. This allows it to detect atom files compressed with the UNIX `compress` command. If the compressed file exists, MODELLER automatically uncompresses it, reads it, and puts it back into the original state after the reading is finished. If the specified file is still not found, the extensions '.atm', '.pdb', '.ent', and '.crd' are tried in this order, without and with extension '.Z', then also with the 'pdb' prefix. This search for the atom file is repeated through all the directories in `ATOM_FILES_DIRECTORY` (directories are separated by ':'), unless input atom filename starts with '/', in which case `ATOM_FILES_DIRECTORY` is neglected. Finally, if still unsuccessful and the file specified by the environment variable `$PDBENT` exists, the coordinate filename (e.g., the 4 character PDB code) is matched to the list of the full PDB filenames in `$PDBENT` (compressed and uncompressed). For example, `$PDBENT` file may be:

```
/disk2/pdb/pdb.pdb.bnl.gov/all_entries/uncompressed_files/pdb1ema.ent
/disk2/pdb/pdb.pdb.bnl.gov/all_entries/uncompressed_files/pdb1hbp.ent
/disk2/pdb/pdb.pdb.bnl.gov/all_entries/uncompressed_files/pdb1gpy.ent
/disk2/pdb/pdb.pdb.bnl.gov/all_entries/uncompressed_files/pdb6gpb.ent
/disk2/pdb/pdb.pdb.bnl.gov/all_entries/uncompressed_files/pdb1fia.ent
etc.
```

Any derivative data that MODELLER may need, including residue solvent accessibilities, hydrogen bonding information, dihedral angles, residue neighbors, etc., are calculated on demand from the atomic coordinates. The most time consuming operation is calculating solvent accessibility, but even this calculation takes less than 1 sec for a 200 residue protein on a Pentium III workstation.

MODELLER stores the filenames of coordinate sets in the alignment arrays. These arrays are used by `COMPARE`, `MAKE_RESTRAINTS`, `MALIGN3D`, `ALIGN2D`, and several other commands. If these filenames

²The 'default' substitution will be phased out because it is a subset of the `DEFAULT` substitution.

do not change when the structures are needed for the second time, the coordinate files are not re-read because they should already be in memory. This creates a problem only when the contents of a structure file changes since it was last read during the current job.

2.1.5 File types

MODELLER uses a number of standard filename extensions to indicate the type of data stored in a file (Table 2.1). The extensions are generally not mandatory, only very helpful.

2.1.6 Format of the command description

For each command, the list of arguments, brief description, and an example are given. Additional background information may be found in Chapter 5. The variable types are described as follows (see also Table ??):

<code><integer : 1></code>	an integer variable or constant
<code><real : 1></code>	a real variable or constant
<code><string : 1></code>	a string variable or constant
<code><logical : 1></code>	a logical variable or constant
<code><integer : 0></code>	a vector of any length with elements <code><integer : 1></code>
<code><integer : N></code>	a vector of N elements <code><integer : 1></code>
<i>etc.</i>	the same for real, string, and logical types

Extension	Description
.top	TOP script with instructions for a MODELLER job
.log	log output produced by a MODELLER run
.ali	alignment or sequences in the PIR format
.pap	alignment or sequences in the PAP format
.aln	alignment or sequences in the QUANTA format
.aln	alignment or sequences in the INSIGHTII format
.seq, .chn	sequence(s) in the PIR alignment format
.cod	list of sequence codes
.grp	list of families in PDB
.atm, .pdb, .ent	atom coordinates in the PDB or GRASP format
.crd	atom coordinates in the CHARMM format
.fit.pdb	fitted protein structures in the PDB format
.ini	initial MODELLER model
.B*	MODELLER model in the PDB format
.D*	the progress of optimization
.BL*	MODELLER model in the PDB format, in loop modeling
.DL*	the progress of optimization, in loop modeling
.IL*	initial MODELLER model, in loop modeling
.V*	violations profile
.E*	energy profile
.rsr	restraints in MODELLER or USER format
.sch	schedule file for the variable target function optimization
.mat	matrix of pairwise protein distances from an alignment
.mat	matrix of pairwise residue type-residue type distance scores
.sim.mat	matrix of pairwise residue type-residue type similarity scores
.lib	various MODELLER libraries
.psa	residue solvent accessibilities
.sol	atomic solvent accessibilities
.ngh	residue neighbors
.dih	mainchain and sidechain dihedral angles
.ssm	secondary structure assignment
.var	sequence variability profile from multiple alignment
.asgl	data for plotting by ASGL

Table 2.1: *List of file types.*

2.2 Stereochemical parameters and molecular topology

All molecular modeling programs generally need to know what are the atoms in all residue types, what are the atom pairs that are covalently bonded to each other (*i.e.*, molecular topology), and what are the ideal bond lengths, angles, dihedral angles, and improper dihedral angles (*i.e.*, internal coordinates and stereochemical restraints). For a given MODEL, these data are constructed mostly from information in the residue topology and parameter libraries. This section describes the commands for reading and writing parameter and residue topology libraries, and for generating, patching, and mutating molecular topology.

2.2.1 Modeling residues with non-existing or incomplete entries in the topology and parameter libraries

Defining new residue types is generally one of the more painful areas in developing and using a molecular modeling program. MODELLER has two quick-and-dirty solutions described in the next two sections that are often sufficient for comparative modeling involving new residue types. On the other hand, if you are willing to spend some time and define a new entry or complete an incomplete entry in the residue topology or parameter libraries, see the FAQ Section 1.9, Question 17.

Residues with defined topology, but with missing parameters

The parameter library is used by the **MAKE_RESTRAINTS** command to construct bond, angle, dihedral angle, improper dihedral angle, and non-bonded Lennard-Jones restraints. If some parameters for these restraints are missing, they are guessed on the fly from the current Cartesian coordinates of the MODEL. Thus, when there are missing parameters, the MODEL coordinates must be defined before calling **MAKE_RESTRAINTS**. The coordinates can be defined by the **BUILD_MODEL** command (from the IC entries in the residue topology library), by the **READ_MODEL** command (from an existing coordinate file for MODEL), or by the **TRANSFER_XYZ** command (from template coordinate files aligned with MODEL). The bonds, angles, and improper dihedral angles are restrained by a harmonic potential with the mean equal to the value in the current structure and a force constant typical for chemical bonds, angles, and improper dihedral angles, respectively. The dihedral angles are restrained by a tri-modal cosine term with the mean equal to the angle in the current structure. A message detailing MODELLER's improvisation is written to the log file.

Block (BLK) residues with undefined topology and parameters

The second relatively easy way of dealing with missing entries in the residue topology and/or parameters libraries is to use a "block" residue. These residues are restrained more or less as rigid bodies to the conformation of the equivalent residue(s) in the template(s). No chemical information is used. The template residues can themselves be defined as block residues. The symbol for the block residues is 'BLK' in the four- and three-letter codes and '.' in the single-letter code. The atoms in a BLK residue include all uniquely named atoms from the equivalent residues in all the templates. The atom type of all BLK atoms is the CHARMM type 'undf'. The IUPAC atom names (as opposed to the atom types) are the same as in the templates. The 'undf' atom type for all BLK atoms facilitates using the **PICK_ATOMS** command for generating restraints on the 'BLK' residues.

The 'undf' atoms are treated differently from the other atoms during preparation of dynamic restraints: No pairs of intra-BLK atoms are put on the dynamic non-bonded list. Only the "inter-BLK" atom pairs and "BLK-other" atom pairs are considered for the dynamic non-bonded restraints. The radius of all block atoms is obtained from the \$RADII_LIB library using the block atom names (as written out to a PDB file), not the 'undf' atom type. All intra-BLK and inter-residue BLK restraints other than the non-bonded restraints have to be derived separately and explicitly by **MAKE_RESTRAINTS** command using **RESTRAINT_TYPE = 'distance'**. See script `scripts/_homcsr.top` for the routine that makes block restraints for comparative modeling by the 'model' script. Lennard-Jones, electrostatic, and general non-bonded spline terms involving 'undf' atoms are ignored by MODELLER.

For an example of how to use block residues, see the FAQ Section 1.9, Question 16.

2.2.2 READ_RESTYP_LIB — read residue type library

Options:

```
RESTYP_LIB_FILE = <string : 1>          '$(LIB)/restyp.lib'  residue type library
```

Description: This command reads residue types from the residue type library specified by variable RESTYP_LIB_FILE.

See the FAQ Section 1.9, Question 17 for the format of this file. MODELLER reads the default residue type library during startup; this command can be used to read residue type definitions for new residue types defined by the user without editing the default residue type library.

Example:

```
# Example for: READ_RESTYP_LIB

# This will read again a user specified residue type library, perhaps
# to read in the new user-defined residue types.

# Just read the default file again for this example:
READ_RESTYP_LIB RESTYP_LIB_FILE = '$(LIB)/restyp.lib'
```

2.2.3 READ_TOPOLOGY — read residue topology library

Options:

```
FILE = <string : 1>          'default'          partial or complete filename
DIRECTORY = <string : 1>     ''                directory list (e.g., 'dir1:dir2:dir3:./:/')
ADD_TOPOLOGY = <logical : 1> off              whether to add new residue topologies to existing
                                                ones
```

Description: This command reads residue topologies from the topology library, such as the CHARMM 22 topology file [?] (it also reads which subsets of atoms correspond to each TOPOLOGY_MODEL from library 'MODELS_LIB'). This file must include atomic connectivities of residues and patching residues, and the internal coordinates for minimum energy residue conformations. Patching residues modify residues; for example, N-terminus, C-terminus and disulfide bonds are defined by patching the original topology. This information is used for generating the molecular topology and possibly for calculating an initial conformation. The default topology for comparative modeling by MODELLER includes only non-hydrogen atoms (TOPOLOGY_MODEL = 3). To define your entries in the topology library, see the FAQ Section 1.9, Questions 17 and 18.

If ADD_TOPOLOGY is on, the new residue topologies are added to the existing residue topologies, otherwise the new topology file replaces the old one. If the topology for a residue is duplicated only the last definition is kept.

Not all the features of the CHARMM 22 topology library are implemented in MODELLER, although a CHARMM file should be read in successfully. A variety of topology files for different kinds of models can be prepared by the MAKE_TOPOLOGY_MODEL command.

The filename for the library is DIRECTORY/FILE.

Example: See PATCH command.

2.2.4 READ_PARAMETERS — read parameters library

Options:

```
FILE = <string : 1>          'default'          partial or complete filename
```

DIRECTORY = <code><string : 1></code>	''	directory list (e.g., 'dir1:dir2:dir3:./:')
ADD_PARAMETERS = <code><logical : 1></code>	off	whether to add new parameters to existing ones

Description: This command reads the parameters from the parameter library, such as the CHARMM 22 parameter file for proteins with all atoms [?]. This file contains the values for bond lengths, angles, dihedral angles, improper dihedral angles, and non-bonded interactions. MODELLER relies on slightly modified CHARMM-22 parameters to reproduce the protein geometry in the MODELLER environment. For example, for the default non-hydrogen atoms model, the ω dihedral angle restraints are stronger than the original CHARMM 22 values which apply to the all-hydrogen model. For a sparse discussion of the parameter library, see the FAQ Section 1.9, Question 17.

If ADD_PARAMETERS is on, the new parameters are added to the existing parameter list, otherwise the contents of the new parameter file replaces the old one.

The filename for the library is DIRECTORY/FILE.

Example: See PATCH command.

2.2.5 READ_ATOM_CLASS — read classification of atom types

Options:

ATOM_CLASSES_FILE = <code><string : 1></code>	'\$(LIB)/atmcls-melo.lib'	library with atom class definitions for MODELLER non-bonded restraints
---	---------------------------	--

Description: This command reads a MODELLER classification of atom types from file ATOM_CLASSES_FILE. This particular atom type classification is used for calculation of the special non-bonded terms other than the soft-sphere, Lennard-Jones or Coulomb terms (for which the CHARMM atom type classification is used). These terms are usually the statistical potentials of mean force described by non-bonded spline restraints, including single body and two body terms. The default atom classification is read during MODELLER initialization.

Example:

```
# Example for: READ_ATOM_CLASSES

# This will read an atom classification for non-bonded statistical potentials
# of mean force.

READ_ATOM_CLASSES ATOM_CLASSES_FILE = '$(LIB)/atmcls-melo.lib'
```

2.2.6 GENERATE_TOPOLOGY — generate MODEL topology

Options:

ADD_SEGMENT = <code><logical : 1></code>	off	whether to add the new segments to the list of segments
PATCH_DEFAULT = <code><logical : 1></code>	on	whether to do default NTER and CTER patching
ALIGN_CODES = <code><string : 0></code>	'all'	codes of proteins in the alignment
SEQUENCE = <code><string : 1></code>	'undefined'	protein code in the alignment whose topology is constructed
ATOM_FILES = <code><string : 0></code>	''	complete or partial atom filenames
ATOM_FILES_DIRECTORY = <code><string : 1></code>	'./'	input atom files directory list (e.g., 'dir1:dir2:dir3:./:')
WATER_IO = <code><logical : 1></code>	off	whether to read water coordinates

HETATM_IO = \langle logical : 1 \rangle	off	whether to read HETATM coordinates
HYDROGEN_IO = \langle logical : 1 \rangle	off	whether to read hydrogen coordinates
TOPOLOGY_MODEL = \langle integer : 1 \rangle	3	selects topology library: 1–9

Requirements: topology and parameter libraries

Description: This command calculates MODEL’s covalent topology (*i.e.*, atomic connectivity) and internal coordinates, and assigns CHARMM atom types, MODELLER atom types for non-bonded spline restraints, atomic charges, and atomic radii.

If a protein with code SEQUENCE is found in the current alignment (codes of proteins in the current alignment are stored in ALIGN_CODES), this protein’s topology is calculated. If no SEQUENCE entry exists or if the alignment does not exist, the sequence of the MODEL is used. If the MODEL does not exist, an error is reported. The MODEL can be read in from an atomic coordinates file with the READ_MODEL command.

The new sequence is added to the list of segments of the MODEL if ADD_SEGMENT is on, otherwise this list is initiated.

A sequence in the alignment can use any non-patching residue listed in the single-character code column of the \$RESTYP_LIB library ('modlib/restyp.lib'). Examples of non-standard residue types include water ('w'), zinc ('z'), calcium ('3'), heme ('h'), and many others. Patching residues must not be used here, but with the subsequent PATCH commands. Unrecognized residues are ignored. A special allowed residue type is the chain break '/'. This can be used to construct a protein that consists of several chains separated by chain breaks. Chain breaks before a non-standard residue type (there are 23 standard residue types, including '-', 'Asx' and 'Glx') are inserted automatically and do not have to be specified explicitly in the sequence.

The GENERATE_TOPOLOGY command generates only the topology of the MODEL, not its Cartesian coordinates; the Cartesian coordinates are assigned by the BUILD_MODEL, TRANSFER_XYZ, or READ_MODEL commands.

In general, the GENERATE_TOPOLOGY command has to be executed before any energy commands (ENERGY, OPTIMIZE, PICK_HOT_ATOMS). The reason is that reading the Cartesian coordinates by the READ_MODEL command does not generate all the data usually needed for energy evaluation. However, if the order and number of atoms in the input file correspond exactly to the order and number of atoms implied by the restraint atom indices and if you are not using dynamic restraints that rely on non-existing data, such as bond, angle, and dihedral angle lists, atomic charges, radii, Lennard-Jones parameters, MODELLER atom types, or CHARMM atom types (which are used to determine the atomic radii), it is sufficient to do only READ_MODEL and omit GENERATE_TOPOLOGY before the energy commands. In short, if you use static restraints alone and if the atom file has the atoms in the correct order, you do not have to call GENERATE_TOPOLOGY before calculating energy.

The variables ATOM_FILES, ATOM_FILES_DIRECTORY, WATER_IO, HETATM_IO, HYDROGEN_IO, and TOPOLOGY_MODEL are necessary only when the 'BLOCK' residues are present in the sequence whose topology is generated. In that case, the template PDB files are read in.

Example: See PATCH command.

2.2.7 PATCH — patch MODEL topology

Options:

RESIDUE_IDS = \langle string : 0 \rangle	''	identifiers of the patched residues
RESIDUE_TYPE = \langle string : 1 \rangle	'undefined'	patching residue type
TOPOLOGY_MODEL = \langle integer : 1 \rangle	3	selects topology library: 1–9

Description: This command uses a CHARMM patching residue to patch the topology of the MODEL. CHARMM patch rules are observed.

RESIDUE_TYPE is the type of the patching residue (PRES entry in the topology library), such as 'DISU', 'NTER', 'CTER', *etc.* You do not have to apply explicitly the N- and C-terminal patches to protein chains because the 'NTER' and 'CTER' patches are applied automatically to the appropriate residue types at the termini of each chain at the end of each **GENERATE_TOPOLOGY** command.

RESIDUE_IDS are residue identifiers of the patched residues (Section 2.4.1). The first residue is the patched residue 1, the second residue is the patched residue 2, *etc.*; for example, the 'DISU' patching residue has two patched Cys residues while the 'ACE' patching residue has only one patched residue. The order of the residue identifiers here has to match the definition of the patching residue in the topology library.

It is not allowed to patch an already patched residue. Since the N- and C-terminal residues of each chain are automatically patched with the 'NTER' and 'CTER' patching residues, respectively, a user who wants to patch the N- or C-terminal residues with other patches, should turn the default patching off before executing **GENERATE_TOPOLOGY**. This is achieved by **SET PATCH_DEFAULT = off**.

Example:

```
# Example for: PATCH, READ_TOPOLOGY, READ_PARAMETERS

# This will define a CYS-CYS disulfide bond between residues 3 and 22.

READ_TOPOLOGY  FILE = '$(LIB)/top_heav.lib'
READ_PARAMETERS FILE = '$(LIB)/par.lib'

# Read the sequence:
READ_MODEL FILE = '1fas'
# have two copies of the sequence in the alignment, for TRANSFER_XYZ later:
SEQUENCE_TO_ALI_ATOM_FILES = '1fas', ALIGN_CODES = '1fas'
SEQUENCE_TO_ALI_ADD_SEQUENCE = on, ATOM_FILES = ATOM_FILES '1fas.ini', ;
                        ALIGN_CODES = ALIGN_CODES '1fas-ini'
GENERATE_TOPOLOGY SEQUENCE = '1fas-ini'

# Create the disulfide bond:
PATCH RESIDUE_TYPE = 'DISU', RESIDUE_IDS = '3' '22'

# Get MODEL's coordinates from the template, using the alignment (1:1 here):
TRANSFER_XYZ
# Calculate missing coordinates using internal coordinates:
BUILD_MODEL INITIALIZE_XYZ = off

# Create the stereochemical restraints
MAKE_RESTRAINTS RESTRAINT_TYPE = 'stereo'

# Calculate the energy to test the disulfide:
ENERGY
```

2.2.8 PATCH_SS_TEMPLATES — guess MODEL disulfides from templates

Options:

ALIGN_CODES = <string: 0>	'all'	codes of proteins in the alignment
ATOM_FILES = <string: 0>	''	complete or partial atom filenames
ATOM_FILES_DIRECTORY = <string: 1>	'./'	input atom files directory list (e.g., 'dir1:dir2:dir3:./:/')
WATER_IO = <logical: 1>	off	whether to read water coordinates

HETATM_IO = \langle logical:1 \rangle	off	whether to read HETATM coordinates
HYDROGEN_IO = \langle logical:1 \rangle	off	whether to read hydrogen coordinates
TOPOLOGY_MODEL = \langle integer:1 \rangle	3	selects topology library: 1-9

Requirements: alignment

Output: DISTANCE_ATOMS

Description: This command defines and patches disulfide bonds in the MODEL using an alignment of the MODEL sequence with one or more template structures. The MODEL sequence has to be the last sequence in the alignment. The template structures are all the other proteins in the alignment. All Cys-Cys pairs in the target sequence that are aligned with at least one template disulfide are defined as disulfide bonds themselves. The covalent connectivity is patched accordingly.

If no alignment exists, a default 1:1 alignment is constructed. Variable ATOM_FILES can be used to specify template structures.

This command should be run after **GENERATE_TOPOLOGY** and before **MAKE_RESTRAINTS** to ensure that the disulfides are restrained properly by the bond length, angle, and dihedral angle restraints and that no SG-SG non-bonded interactions are applied.

The disulfide bond, angle and dihedral angle restraints have their own physical restraint type separate from the other bond, angle and dihedral angle restraints (Table 2.4).

DISTANCE_ATOMS becomes CA SG.

Example:

```
# Example for: PATCH_SS_TEMPLATES and PATCH_SS_MODEL

# This will patch CYS-CYS disulfide bonds using disulfides in aligned templates:

SET OUTPUT_CONTROL = 1 1 1 1 1

READ_TOPOLOGY FILE = '$(LIB)/top_heav.lib'
READ_PARAMETERS FILE = '$(LIB)/par.lib'

# Read the sequence, calculate its topology, and coordinates:
READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = '2ctx' '2abx'
# Superpose the two template structures without changing the alignment.
# This is for TRANSFER_XYZ to work properly. It relies on not reading
# the atom files again before TRANSFER_XYZ.
MALIGN3D FIT = off # This is for TRANSFER_XYZ to work properly.
READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = ALIGN_CODES '1fas'
GENERATE_TOPOLOGY SEQUENCE = '1fas'
TRANSFER_XYZ
BUILD_MODEL INITIALIZE_XYZ = on
WRITE_MODEL FILE = '1fas.noSS'
# Create the disulfide bonds using equivalent disulfide bonds in templates:
PATCH_SS_TEMPLATES

# Create the stereochemical restraints
MAKE_RESTRAINTS RESTRAINT_TYPE = 'stereo'

# Calculate energy to test the disulfide restraints (bonds, angles, dihedrals):
ENERGY
```

```

READ_MODEL FILE = '1fas.noSS'
# Create the disulfide bonds guessing by coordinates
PATCH_SS_MODEL

# Create the stereochemical restraints
MAKE_RESTRAINTS RESTRAINT_TYPE = 'stereo'

# Calculate energy to test the disulfide restraints (bonds, angles, dihedrals):
ENERGY

```

2.2.9 PATCH_SS_MODEL — guess MODEL disulfides from model structure

Options:

```

TOPOLOGY_MODEL = <integer : 1>      3          selects topology library: 1–9

```

Requirements: model

Description: This command defines and patches disulfide bonds in MODEL using MODEL's current structure. A disulfide bridge is declared between all pairs of Cys residues whose SG–SG distances are less than 2.5Å. The covalent connectivity is patched accordingly.

This command should be run after **READ_MODEL** and before optimization to ensure that the disulfides are fixed properly and that no SG–SG non-bonded interactions are applied.

TOPOLOGY_MODEL is needed to make sure the correct atomic radii are used in CYS–CYS patching.

Example: See **PATCH_SS_TEMPLATES** command.

2.2.10 MUTATE_MODEL — mutate selected MODEL residues

Options:

```

RESIDUE_TYPE = <string : 1>          'undefined'      new residue type

```

Description: This command mutates the selected residues of the MODEL to the type specified by RESIDUE_TYPE. CHARMM 4-character residue type names are used (see library file \$RESTYP_LIB). To select the residues for mutation, use **PICK_ATOMS** command. All the residues with at least one atom in the selected set 1 of atoms are mutated. To produce mutants, employ this command with **SEQUENCE_TO_ALI** and **WRITE_ALIGNMENT**. It is usually necessary to write the mutated sequence out and read it in before proceeding, because not all sequence related information about MODEL is changed by this command (*e.g.*, internal coordinates, charges, and atom types and radii are not updated).

Example:

```

# Example for: MUTATE_MODEL

# This will read a PDB file, change its sequence a little, build new
# coordinates for any of the additional atoms using only the internal
# geometry, and write the mutant PDB file. It can be seen as primitive,

```

```

# but rapid comparative modeling for substitution mutants. For insertion
# and deletion mutants, follow the standard comparative modeling procedure.

# Read the topology library with non-hydrogen atoms only:
READ_TOPOLOGY FILE = '$(LIB)/top_heav.lib', TOPOLOGY_MODEL = 3
# To produce a mutant with all hydrogens, uncomment this line:
# READ_TOPOLOGY FILE = '$(LIB)/top.lib', TOPOLOGY_MODEL = 1

# Read the CHARMM parameter library:
READ_PARAMETERS FILE = '$(LIB)/par.lib'

# Read the original PDB file and copy its sequence to the alignment array:
READ_MODEL FILE = '1fas'
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ATOM_FILES = '1fas', ALIGN_CODES = '1fas'

# Select the residues to be mutated: in this case all ASP residues:
PICK_ATOMS RES_TYPES = 'ASP'

# The second example is commented out; it selects residues '1' and '10'.
# SET SELECTION_SEARCH = 'SEGMENT', SELECTION_FROM = 'ALL'
# PICK_ATOMS SELECTION_SEGMENT = '1' '1', SELECTION_STATUS = 'INITIALIZE'
# PICK_ATOMS SELECTION_SEGMENT = '10' '10', SELECTION_STATUS = 'ADD'

# Mutate the selected residues into HSD residues (neutral HIS):
MUTATE_MODEL RESIDUE_TYPE = 'HSD'

# Add the mutated sequence to the alignment arrays (it is now the second
# sequence in the alignment):
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ALIGN_CODES = ALIGN_CODES '1fas-1'

# Generate molecular topology for the mutant:
GENERATE_TOPOLOGY SEQUENCE = '1fas-1'

# Transfer all the coordinates you can from the template native structure
# to the mutant (this works even if the order of atoms in the native PDB
# file is not standard):
TRANSFER_XYZ

# Build the remaining unknown coordinates for the mutant:
BUILD_MODEL INITIALIZE_XYZ = off

# Write the mutant to a file:
WRITE_MODEL FILE = '1fas-1.atm'

```

2.2.11 MAKE_TOPOLOGY_MODEL — make a subset topology library

Options:

TOPOLOGY_MODEL = {integer : 1} 3 selects topology library: 1-9

Description: This command makes a residue topology library from the most detailed CHARMM topology library, which contains all atoms, including all hydrogens (corresponding to TOPOLOGY_MODEL = 1). There are currently ten residue topologies, all of which are defined in library \$MODELS_LIB. For example, the default

non-hydrogen atom topology is selected by `TOPOLOGY_MODEL = 3`. For each `TOPOLOGY_MODEL` and residue type, the `$MODELS_LIB` library lists those atoms in the full atom set that are part of the specified topology. This command works by deleting all the entries that contain non-existing atoms from the original topology file. One must carefully test topology files produced in this way. Library `$RADII_LIB` must specify atomic radii for each atom in each residue type for each topology model. `TOPOLOGY_MODEL` must be an integer from 1 to 10. For more information about the topology library, see the FAQ Section 1.9, Questions 17 and 18.

Example:

```
# Example for: MAKE_TOPOLOGY_MODEL, WRITE_TOPOLOGY_MODEL

# This creates a topology library for heavy atoms from the
# CHARMM all-atom topology library:

# Read CHARMM all-atom topology library:
READ_TOPOLOGY FILE = '${LIB}/top.lib'

# Keep only heavy atoms (TOPOLOGY_MODEL = 3)
MAKE_TOPOLOGY_MODEL TOPOLOGY_MODEL = 3

# Write the resulting topology library to a new file:
WRITE_TOPOLOGY_MODEL FILE = 'top_heav.lib'
```

2.2.12 WRITE_TOPOLOGY_MODEL — write residue topology library

Options:

<code>FILE = <string : 1></code>	<code>'default'</code>	partial or complete filename
<code>OUTPUT_DIRECTORY = <string : 1></code>	<code>''</code>	output directory

Description: This command writes a residue topology library to the specified file. It is usually used after `MAKE_TOPOLOGY_MODEL`.

Example: See `MAKE_TOPOLOGY_MODEL` command.

2.3 Handling of atomic coordinates

2.4 Comparison and searching of sequences and structures

This section describes the format of the alignment file and commands for reading, writing, making, analyzing and using the alignments of sequences and structures (pairwise and multiple). It also includes a description of the command for searching a sequence database. For the underlying dynamic programming methods see Section 5.1.

2.4.1 Alignment file format

The preferred format for comparative modeling is related to the PIR database format:

```
C; A sample alignment in the PIR format; used in tutorial
>P1;5fd1
structureX:5fd1:1    : :106  : :ferredoxin:Azotobacter vinelandii: 1.90: 0.19
AFVVTDNCKIKCKYTDCVEVCPVDCFYEGPNFLVIHPDECIDCALCEPECPAQAIFSEDEVPEDMQEFIQLNAELA
EVWPNITEKKDPLPDAEDWDGKGLQHLER*
>P1;1fdx
sequence:1fdx:1    : :54   : :ferredoxin:Peptococcus aerogenes: 2.00:-1.00
AYVINDSC--IACGACKPECPVNI IQGS--IY AIDADSCIDCGSCASVCPVGAPNPED-----
-----*
```

The first line of each sequence entry specifies the protein code after the >P1; line identifier. The line identifier must occur at the beginning of the line. For example, 1fdx is the protein code of the first entry in the alignment above. The protein code corresponds to the ALIGN_CODES variable.

The second line of each entry contains information necessary to extract atomic coordinates of the segment from the original PDB coordinate set. The fields in this line are separated by colon characters, ':'. The fields are as follows:

- Field 1: A specification of whether or not 3D structure is available and of the type of the method used to obtain the structure (**structureX**, X-ray; **structureN**, NMR; **structureM**, model; **sequence**, sequence). Only **structure** is also a valid value.
- Field 2: The PDB code. While the protein code in the first line of an entry, which is used to identify the entry, must be unique for all proteins in the file, the PDB code in this field, which is used to get structural data, does not have to be unique. It is a good idea to use the PDB code with an optional chain identifier as the protein code. The PDB code corresponds to the ATOM_FILES variable and can also contain the full atom filename, directory included.
- Fields 3–6: The residue identifiers (see below) for the first (fields 3–4) and last residue (fields 5–6) of the sequence in the subsequent lines. There is no need to edit the coordinate file if a contiguous sequence of residues is required — simply specify the beginning and ending residues of the required contiguous region of the chain. If the beginning residue is not found, no segment is read in. If the ending residue identifier is not found in the coordinate file, the last residue in the coordinate file is used. By default, the whole file is read in.
 The unspecified beginning and ending residue numbers and chain id's for a **structure** entry in an alignment file are taken automatically from the corresponding atom file, if possible. The first matching sequence in the atom file that also satisfies the explicitly specified residue numbers and chain id's is used. A residue number is not specified when a blank character or a dot, '.', is given. A chain id is not specified when a dot, '.', is given. This slight difference between residue and chain id's is necessary because a blank character is a valid chain id.
- Field 7: Protein name. Optional.
- Field 8: Source of the protein. Optional.
- Field 9: Resolution of the crystallographic analysis. Optional.
- Field 10: R-factor of the crystallographic analysis. Optional.

A residue identifier consists of a residue number and an optional chain identifier. They must be separated by a colon, ':'. For example, '10I:A' is residue number '10I' in chain 'A', and '6' or '6:' is residue number '6' in a chain without a name. Free format can be used, that is the blank characters are ignored. The residue number is a string of up to 5 characters long, as found in the PDB atom file and consists of the PDB residue number proper (22X,A4 in the PDB ATOM record) and PDB residue insertion code (26X, A1). The chain identifier is a single character, as found in the PDB atom file (21X,A1). A string containing '@' will match any residue number and chain id. For example, '@:A' is the first residue in chain 'A' and '@:@' is the first residue in the coordinate file. The same residue identifier format is used in several other TOP variables used for residue specification (*e.g.*, MODEL_SEGMENT, SELECTION_SEGMENT).

When an alignment file is used in conjunction with structural information, the first two fields must be filled in, the rest of them can be empty or even missing entirely. If the alignment is not used in conjunction with structural data, all but the first field can be empty. This means that in comparative modeling, the template structures must have at least the first two fields specified while the target sequence must only have the first field filled in. Thus, a simple second line of an entry in an alignment file in the 'PIR' format is

```
structure:pdb_file:.....
```

This entry will result in reading from PDB file `pdb_file` the structure segment corresponding to the sequence in the subsequent lines of the alignment entry.

The fields that do not exist are assigned blank values. Thus,

```
structure:pdb_file
```

is equivalent to

```
structure:pdb_file: : : : : : :
```

which will achieve what was probably intended (read in the structure segment from file `pdb_file` that corresponds to the sequence in the subsequent lines of the alignment entry) only if the chain id is a blank character.

Each sequence must be terminated by the terminating character, '*'.

When the first character of the sequence line is the terminating character, '*', the sequence is obtained from the specified PDB coordinate file (Section 2.1.4).

Chain breaks are indicated by '/'. There should not be more than one chain break character to indicate a single chain break (use gap characters instead, '-'). All residue types specified in \$RESTYP_LIB, but not patching residue types, are allowed; there are on the order of 100 residue types specified in the \$RESTYP_LIB library. To add your own residue types to this library, see Section 1.9, Question 17.

The alignment file can contain any number of blank lines between the protein entries. Comment lines can occur outside protein entries and must begin with the identifiers 'C;' or 'R;' as the first two characters in the line.

An alignment file is also used to input non-aligned sequences.

The best way to generate initial alignment files containing PDB sequences, which can later be edited by hand, is to follow this example:

```
# Specify the PDB and protein codes in the alignment:
SET ATOM_FILES = '1fdx' '5fd1', ALIGN_CODES = '1fdx' '5fd1'
READ_MODEL FILE = '1fdx', MODEL_SEGMENT = '@:@' 'X:X' # Read the whole 1fdx atom file
SEQUENCE_TO_ALI # Copy the residues to the alignment array
READ_MODEL FILE = '5fd1', MODEL_SEGMENT = '1:' '63:' # Read 5fd1 atom file from 1-63
SEQUENCE_TO_ALI ADD_SEQUENCE = on # Add this segment to the alignment array
MALIGN GAP_PENALTIES = -500 -300 # align them by sequence
WRITE_ALIGNMENT FILE = 'fer1-seq.ali'
MALIGN3D GAP_PENALTIES = 0.0 2.0 # align them by structure
CHECK_ALIGNMENT # check the alignment for its suitability for modeling
WRITE_ALIGNMENT FILE = 'fer1.ali'
```

2.4.2 READ_ALIGNMENT — read sequences and/or their alignment

Options:

FILE = <code><string : 1></code>	'default'	partial or complete filename
DIRECTORY = <code><string : 1></code>	''	directory list (e.g., 'dir1:dir2:dir3:./:/')
ALIGN_CODES = <code><string : 0></code>	'all'	codes of proteins in the alignment
ALIGNMENT_FORMAT = <code><string : 0></code>	'PIR'	format of the alignment file: 'PIR' 'PAP' 'QUANTA' 'INSIGHT' 'FASTA'
REMOVE_GAPS = <code><logical : 1></code>	on	whether to remove all-gap positions in input alignment
ADD_SEQUENCE = <code><logical : 1></code>	off	whether to add the new sequences to the existing alignment
STOP_ON_ERROR = <code><integer : 1></code>	1	whether to stop on error

Output: MODELLER_STATUS = `<integer : 1>`, NUMB_OF_SEQUENCES, ALIGN_CODES

Description: This command reads the sequence(s) and/or their alignment from a text file. Only sequences with the specified codes are read in; ALIGN_CODES = 'all' can be used to read all sequences.

There are several alignment formats:

1. The 'PIR' format resembles that of the PIR sequence database. It is described in Section 2.4.1 and is used for comparative modeling because it allows for additional data about the proteins that are useful for automated access to the atomic coordinates.
2. The 'FASTA' format resembles the 'PIR' format but has a missing second 'comment' line and a missing star at the end of each sequence.
3. The 'PAP' format is nicer to look at but contains less information and is not used by other programs. When used in conjunction with PDB files, the PDB files must contain exactly the residues in the sequences in the 'PAP' file; *i.e.*, it is not possible to use only a segment of a PDB file. In addition, the 'PAP' protein codes must be expandable into proper PDB atom filenames, as described in Section 2.1.4. The protein sequence can now start in any column (this was limited to column 11 before release 5).
4. The 'QUANTA' format can be used to communicate with the QUANTA program. You are not supposed to mix 'QUANTA' format with any other format because the 'QUANTA' format contains residue numbers which do not occur in the other formats and are difficult to guess correctly.
5. The 'INSIGHT' format is very similar to the 'PAP' format and can sometimes be used to communicate with the INSIGHTII program. When used in conjunction with PDB files, the same rules as for the 'PAP' format apply.

If REMOVE_GAPS = on, positions with gaps in all selected sequences are removed from the alignment.

If ADD_SEQUENCE is on, the new sequences are added to the current ones, otherwise the old sequences are deleted.

Example:

```
# Example for: READ_ALIGNMENT, WRITE_ALIGNMENT,
#             READ_ALIGNMENT2, WRITE_ALIGNMENT2,
#             CHECK_ALIGNMENT

# Read an alignment, write it out in the 'PAP' format, and
# check the alignment of the N-1 structures as well as the
# alignment of the N-th sequence with each of the N-1 structures.

SET OUTPUT_CONTROL = 1 1 1 1 0
```



```

READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = 'all'
WRITE_ALIGNMENT FILE = 'toxin.pap', ALIGNMENT_FORMAT = 'PAP'
WRITE_ALIGNMENT FILE = 'toxin.fasta', ALIGNMENT_FORMAT = 'FASTA'
CHECK_ALIGNMENT

```

2.4.3 READ_ALIGNMENT2 — read 2nd alignment

Options:

FILE = <code><string : 1></code>	'default'	partial or complete filename
DIRECTORY = <code><string : 1></code>	''	directory list (e.g., 'dir1:dir2:dir3:./:/')
ALIGN_CODES2 = <code><string : 0></code>	'all'	align codes for alignment2
ALIGNMENT_FORMAT = <code><string : 0></code>	'PIR'	format of the alignment file: 'PIR' 'PAP' 'QUANTA' 'INSIGHT' 'FASTA'
REMOVE_GAPS = <code><logical : 1></code>	on	whether to remove all-gap positions in input alignment
STOP_ON_ERROR = <code><integer : 1></code>	1	whether to stop on error

Output: MODELLER.STATUS = `<integer : 1>`

Description: This command reads the sequences and/or their alignment from a text file into the second alignment array. The two alignments can be compared by the **COMPARE_ALIGNMENTS** command.

Example: See **READ_ALIGNMENT** command.

2.4.4 CHECK_ALIGNMENT — check alignment for modeling

Options:

ATOM_FILES = <code><string : 0></code>	''	complete or partial atom filenames
ALIGN_CODES = <code><string : 0></code>	'all'	codes of proteins in the alignment

Description: This command evaluates an alignment to be used for comparative modeling. It uses two criteria:

First, it checks the alignment of the template structures (all but the last entry in the alignment): For each pairwise superposition of the templates, it reports those equivalent pairs of C_{α} atoms that are more than 6Å away from each other. Such pairs are almost certainly misaligned. The pairwise superpositions are done using the C_{α} atoms and the given alignment.

Second, the command checks the alignment of the target sequence (the last entry in the alignment) with each of the templates: For all consecutive pairs of C_{α} atoms in the target, it calculates the distance between the two equivalent C_{α} atoms in each of the templates. If the distance is longer than 8Å, it is reported. In such a case, the alignment between the template and the target is almost certainly incorrect.

Example:

```

# Example for: READ_ALIGNMENT, WRITE_ALIGNMENT,
#             READ_ALIGNMENT2, WRITE_ALIGNMENT2,
#             CHECK_ALIGNMENT

# Read an alignment, write it out in the 'PAP' format, and
# check the alignment of the N-1 structures as well as the
# alignment of the N-th sequence with each of the N-1 structures.

```

```

SET OUTPUT_CONTROL = 1 1 1 1 0

READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = 'all'
WRITE_ALIGNMENT FILE = 'toxin.pap', ALIGNMENT_FORMAT = 'PAP'
WRITE_ALIGNMENT FILE = 'toxin.fasta', ALIGNMENT_FORMAT = 'FASTA'
CHECK_ALIGNMENT

```

2.4.5 COLOR_ALN_MODEL — color MODEL according to alignment

Description: This command colors MODEL according to a given alignment between MODEL and a sequence. MODEL has to be the first protein in the alignment. The second protein can be any sequence, with or without known structure. The MODEL can be displayed on the screen, colored by 'the fourth parameter' and inspected for the structural context of deletions and insertions. This is useful in optimizing the alignment for comparative modeling. The isotropic temperature factors in MODEL are set as follows:

- 0, for those regions that have residues in both MODEL and the sequence (blue in RASMOL; light green in QUANTA);
- 1, for the two residues that span regions occurring in the sequence but not in MODEL (green in RASMOL; pink in QUANTA);
- 2, regions that occur in MODEL but are deleted from the sequence (red in RASMOL; bright green in QUANTA).

Example:

```

# Example for: COLOR_ALN_MODEL

# Two demos:
#
# 1) Use a given alignment to color a structure according to
#     insertions and deletions in a pairwise alignment.
#
# 2) Superpose two 3D structure and do (1).

# Demo 1:

READ_MODEL FILE = '1nbt'
READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = '1nbt' '1fas', ;
      REMOVE_GAPS = on
COLOR_ALN_MODEL
WRITE_MODEL FILE = '1nbt-1.clr'

# Demo 2:

READ_MODEL FILE = '1nbt', MODEL_SEGMENT = '1:A' '66:A'
SEQUENCE_TO_ALI ALIGN_CODES = '1nbt', ATOM_FILES = ALIGN_CODES
READ_MODEL FILE = '1fas', MODEL_SEGMENT = '1:' '61:'
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ALIGN_CODES = ALIGN_CODES '1fas', ;
      ATOM_FILES = ALIGN_CODES
ALIGN GAP_PENALTIES_1D= -600 -400
MALIGN3D GAP_PENALTIES_3D= 0 3.0
WRITE_ALIGNMENT FILE = 'color_aln_model.pap', ALIGNMENT_FORMAT = 'PAP'

```

```

READ_MODEL FILE = '1nbt', MODEL_SEGMENT = '1:A' '66:A'
COLOR_ALN_MODEL
WRITE_MODEL FILE = '1nbt-2.clr'

```

2.4.6 COMPARE_ALIGNMENTS — compare two alignments

Requirements: READ_ALIGNMENT & READ_ALIGNMENT2

Description: This command compares two pairwise alignments read by the **READ_ALIGNMENT** and **READ_ALIGNMENT2** commands. The alignment of the first sequence with the second sequence in **ALIGNMENT** is evaluated with respect to **ALIGNMENT2**. The numbers are not symmetric; *i.e.*, they will change if the sequences or alignments are swapped. The output in the log file is self-explanatory.

Example:

```

# Example for: COMPARE_ALIGNMENTS, SEQUENCE_TO_ALI

# Compare two alignments of two proteins each. In this case, the first
# alignment is a sequence-sequence alignment and the second alignment
# is a structure-structure alignment.

# ATOM_FILES and ALIGN_CODES have to be set explicitly so that the alignment
# file has this information (ATOM_FILES is copied to the alignment array
# during SEQUENCE_TO_ALI):

SET OUTPUT_CONTROL = 1 1 1 1 0

# Generate and save sequence-sequence alignment:
READ_MODEL FILE = '1fas'
SEQUENCE_TO_ALI ALIGN_CODES = '1fas', ATOM_FILES = ALIGN_CODES
READ_MODEL FILE = '2ctx'
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ALIGN_CODES = ALIGN_CODES '2ctx', ATOM_FILES = ALIGN_CODES
ALIGN GAP_PENALTIES_1D= -600 -400
WRITE_ALIGNMENT FILE = 'toxin-seq.ali'

# Generate and save structure-structure alignment:
ALIGN3D GAP_PENALTIES_3D= 0 2.0
WRITE_ALIGNMENT FILE = 'toxin-str.ali'

SET ADD_SEQUENCE = off

# Compare the two pairwise alignments:
READ_ALIGNMENT FILE = 'toxin-seq.ali', ALIGN_CODES = 'all'
READ_ALIGNMENT2 FILE = 'toxin-str.ali', ALIGN_CODES2 = 'all'
COMPARE_ALIGNMENTS

```

2.4.7 SEQUENCE_TO_ALI — copy MODEL sequence and coordinates to alignment

Options:

```

ADD_SEQUENCE = (logical : 1)      off      whether to add the new sequences to the existing
                                         alignment

```

ALIGN_CODES = <code><string : 0></code>	<code>'all'</code>	codes of proteins in the alignment
ATOM_FILES = <code><string : 0></code>	<code>''</code>	complete or partial atom filenames
OUTPUT_DIRECTORY = <code><string : 1></code>	<code>''</code>	output directory

Output: NUMB_OF_SEQUENCES, ALIGN_CODES, ATOM_FILES

Description: This command copies the sequence and coordinates of the MODEL to the alignment arrays.

If ADD_SEQUENCE is on the sequence is added to the sequences that are already in the alignment arrays, otherwise it becomes the only sequence in those arrays.

When sequence *i* is added, the corresponding elements of ALIGN_CODES and ATOM_FILES are used to set the protein and PDB code fields in the alignment file, respectively.

Example: See COMPARE_ALIGNMENTS command.

2.4.8 WRITE_ALIGNMENT — write sequences and/or their alignment

Options:

FILE = <code><string : 1></code>	<code>'default'</code>	partial or complete filename
OUTPUT_DIRECTORY = <code><string : 1></code>	<code>''</code>	output directory
ATOM_FILES = <code><string : 0></code>	<code>''</code>	complete or partial atom filenames
ALIGN_CODES = <code><string : 0></code>	<code>'all'</code>	codes of proteins in the alignment
ALIGNMENT_FORMAT = <code><string : 0></code>	<code>'PIR'</code>	format of the alignment file: <code>'PIR'</code> <code>'PAP'</code> <code>'QUANTA'</code> <code>'INSIGHT'</code> <code>'FASTA'</code>
ALIGN_BLOCK = <code><integer : 1></code>	<code>1</code>	the last sequence in the first block of sequences
ALIGN_ALIGNMENT = <code><logical : 1></code>	<code>off</code>	writing out an alignment of alignments (for *)
ALIGNMENT_FEATURES = <code><string : 0></code>	<code>'INDICES CONSERVATION'</code>	what alignment features to write out: <code>'ACCURACY'</code> <code>'HELIX'</code> <code>'BETA'</code> <code>'ACCESSIBILITY'</code> <code>'STRAIGHTNESS'</code> <code>'CONSERVATION'</code> <code>'INDICES'</code> <code>'ALL'</code> <code>'GAPS'</code>
CUT_OVERHANGS = <code><logical : 1></code>	<code>off</code>	whether to cut overhangs at OVERHANG residues or not
OVERHANG = <code><integer : 1></code>	<code>0</code>	un-penalized overhangs in protein comparisons
HETATM_IO = <code><logical : 1></code>	<code>off</code>	whether to read HETATM coordinates
WATER_IO = <code><logical : 1></code>	<code>off</code>	whether to read water coordinates
HYDROGEN_IO = <code><logical : 1></code>	<code>off</code>	whether to read hydrogen coordinates
ATOM_FILES_DIRECTORY = <code><string : 1></code>	<code>'./'</code>	input atom files directory list (e.g., <code>'dir1:dir2:dir3:./:/'</code>)

Description: This command writes the whole alignment to a text file.

The 'PAP' format, which corresponds to a relatively nice looking alignment, has several additional formatting options that can be selected by the ALIGNMENT_FEATURES variable. This scalar variable can contain any combination of the following keywords:

- 'INDICES', the alignment position indices;
- 'CONSERVATION', a star for each absolutely conserved position;
- 'ACCURACY', the alignment accuracy indices, scaled between 0–9, as calculated by **ALIGN_CONSENSUS**;
- 'HELIX', average content of helical residues for structures 1 – ALIGN_BLOCK at each position, 0 for 0% and 9 for 100%, as calculated by **ALIGN2D**.

- 'BETA', average content of β -strand residues for structures 1 – ALIGN_BLOCK at each position, 0 for 0% and 9 for 100%, as calculated by **ALIGN2D**.
- 'ACCESSIBILITY', average relative sidechain buriedness for structures 1 – ALIGN_BLOCK, 0 for 0% (100% accessibility) and 9 for 100% (0% accessibility), as calculated by **ALIGN2D**;
- 'STRAIGHTNESS', average mainchain straightness structures 1 – ALIGN_BLOCK at each position 0 for 0% and 9 for 100%, as calculated by **ALIGN2D**.

Options 'HELIX', 'BETA', 'ACCESSIBILITY', and 'STRAIGHTNESS' are valid only after executing command **ALIGN2D**, where the corresponding quantities are defined. They refer to the 3D profile defined for the first ALIGN_BLOCK structures (run **ALIGN2D** with FIT = off to prepare these structural data without changing the input alignment). Similarly, the 'ACCURACY' option is valid only after the **CONSENSUS_ALIGNMENT** command.

ALIGN_ALIGNMENT and ALIGN_BLOCK are used to ensure correct indication of identical alignment positions, depending on whether sequences or two blocks of sequences were aligned: For sequences (ALIGN_ALIGNMENT = off and ALIGN_BLOCK is ignored), a '*' indicating a conserved position is printed where all sequences have the same residue type. For blocks (ALIGN_ALIGNMENT = on and ALIGN_BLOCK indicates the last sequence of the first block), a '*' is printed only where the two blocks have the same order of residue types (there has to be the same number of sequences in both blocks). The blocks option is useful when comparing two alignments, possibly aligned by the **ALIGN** command.

If CUT_OVERHANGS is on, overhangs longer than OVERHANG residues are cut from the alignment. In such a case, the HETATM_IO, WATER_IO, HYDROGEN_IO, and ATOM_FILES_DIRECTORY keywords also apply because the beginning and ending residue numbers for the 'structure' entries in the alignment are renumbered automatically by reading the appropriate atom files.

Example: See **READ_ALIGNMENT** command.

2.4.9 DESCRIBE — describe proteins

Options:

ATOM_FILES = <string: 0>	''	complete or partial atom filenames
ATOM_FILES_DIRECTORY = <string: 1>	'./'	input atom files directory list (e.g., 'dir1:dir2:dir3:./:')
ALIGN_CODES = <string: 0>	'all'	codes of proteins in the alignment

Requirements: [alignment]

Description: This command outputs basic data about the proteins whose atom filenames are specified by ATOM_FILES or ALIGN_CODES. An alternative specification of the proteins to be described can be provided by the alignment in memory; *i.e.*, **READ_ALIGNMENT** followed by **DESCRIBE** will describe all the proteins in the alignment. This command is useful for preparation before comparative modeling because it summarizes disulfides, *cis*-prolines, charges, chain breaks, *etc.* When an alignment is given, results depending only on the amino acid sequences are still written out even if some atom files do not exist.

Example:

```
# Example for: DESCRIBE

# Describe the sequences and structures in the alignment.

READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = '2ctx' '2abx'
DESCRIBE
```

2.4.10 ID_TABLE — calculate percentage sequence identities

Options:

ALIGN_CODES = <code><string : 0></code>	<code>'all'</code>	codes of proteins in the alignment
MATRIX_FILE = <code><string : 1></code>	<code>'family.mat'</code>	the filename of the pairwise distance matrix
OUTPUT_DIRECTORY = <code><string : 1></code>	<code>''</code>	output directory

Requirements: alignment

Description: This command calculates percentage residue identities for all pairs of sequences in the current alignment. The percentage residue identity is defined as the number of identical residues divided by the length of the shorter sequence.

The ALIGN_CODES variable is only used for output, not in calculations, so it does not have to be set.

In addition to the output in the log file, this routine creates file OUTPUT_DIRECTORY/MATRIX_FILE with pairwise sequence distances that can be used directly as the input to the tree making programs of the PHYLIP package, such as KITSCH [?], and also for the **DENDROGRAM** and **PRINCIPAL_COMPONENTS** commands. A more general version of this command, which allows a user specified measure for residue-residue differences is **SEQUENCE_COMPARISON**.

Example:

```
# Example for: ID_TABLE, SEQUENCE_COMPARISON, PRINCIPAL_COMPONENTS, DENDROGRAM

# Pairwise sequence identity between sequences in the alignment.

# Read all entries in this alignment:
READ_ALIGNMENT FILE = 'toxin.ali'

# Calculate pairwise sequence identities:
ID_TABLE MATRIX_FILE = 'toxin_id.mat'

# Calculate pairwise sequence similarities:
SET RR_FILE = '$(LIB)/as1.sim.mat', MAX_GAPS_MATCH = 1
READ_MODEL FILE = '2ctx', MODEL_SEGMENT = '1:' '71:'
SEQUENCE_COMPARISON MATRIX_FILE = 'toxin.mat', VARIABILITY_FILE = 'toxin.var'
WRITE_MODEL FILE = '2ctx.var'

# Do principal components clustering using sequence similarities:
PRINCIPAL_COMPONENTS MATRIX_FILE = 'toxin.mat', FILE = 'toxin.princ'

# Dendrogram in the log file:
DENDROGRAM
```

2.4.11 SEQUENCE_COMPARISON — compare sequences in alignment

Options:

RR_FILE = <code><string : 1></code>	<code>'\$(LIB)/as1.sim.mat'</code>	input residue-residue scoring file
DIRECTORY = <code><string : 1></code>	<code>''</code>	directory list (e.g., <code>'dir1:dir2:dir3:./:/'</code>)
MATRIX_FILE = <code><string : 1></code>	<code>'family.mat'</code>	the filename of the pairwise distance matrix
VARIABILITY_FILE = <code><string : 1></code>	<code>'undefined'</code>	output filename

```

OUTPUT_DIRECTORY = <string : 1>      ''                output directory
ALIGN_CODES = <string : 0>           'all'           codes of proteins in the alignment
MAX_GAPS_MATCH = <integer : 1>       1

```

Description: The pairwise similarity of sequences in the current alignment is evaluated using a user specified residue–residue scores file.

The residue–residue scores, including gap–residue, and gap–gap scores, are read from file `RR_FILE`. The sequence pair score is equal to the average pairwise residue–residue score for all alignment positions that have at most `MAX_GAPS_MATCH` gaps (1 by default). If the gap–residue and gap–gap scores are not defined in `MATRIX_FILE`, they are set to the worst and best residue–residue score, respectively. If `MATRIX_FILE` is a similarity matrix, it is converted into a distance matrix ($x' = -x + x_{\max}$).

The comparison matrix is written in the PHYLIP format to file `MATRIX_FILE`.

The family variability as a function of alignment position is calculated as the RMS deviation of all residue – residue scores at a given position, but only for those pairs of residues that have at most `MAX_GAPS_MATCH` gaps (0, 1, or 2). The variability is written to file `VARIABILITY_FILE`, as is the number of pairwise comparisons contributing to each positional variability.

Example: See `ID_TABLE` command.

2.4.12 DENDROGRAM — clustering

Options:

```

MATRIX_FILE = <string : 1>           'family.mat'    the filename of the pairwise distance matrix

```

Description: This command calculates a clustering tree from the input matrix of pairwise distances. This matrix must be in the PHYLIP format and can be produced by the `ID_TABLE`, `SEQUENCE_COMPARISON`, or `COMPARE` commands. The weighted pair-group average clustering method is used.

The tree is written to the `log` file.

This command is useful for deciding about which known 3D structures are to be used as templates for comparative modeling.

Example: See `ID_TABLE` command.

2.4.13 PRINCIPAL_COMPONENTS — clustering

Options:

```

MATRIX_FILE = <string : 1>           'family.mat'    the filename of the pairwise distance matrix
FILE = <string : 1>                  'default'       output file

```

Description: This command calculates principal components clustering for the input matrix of pairwise distances. This matrix must be in the PHYLIP format and can be produced by the `ID_TABLE`, `SEQUENCE_COMPARISON`, or `COMPARE` commands.

The projected coordinates p and q are written to file `FILE`. The output file can be used with `ASGL` to produce a principal components plot.

This command is useful for deciding about which known 3D structures are to be used as templates for comparative modeling.

Example: See `ID_TABLE` command.

2.4.14 ALIGN — align two (blocks of) sequences

Options:

RR_FILE = <code><string : 1></code>	'\$(LIB)/as1.sim.mat'	input residue-residue scoring file
DIRECTORY = <code><string : 1></code>	''	directory list (e.g., 'dir1:dir2:dir3.:/:')
GAP_PENALTIES_1D = <code><real : 2></code>	-900 -50	gap creation and extension penalties for sequence/sequence alignment
ALIGN_BLOCK = <code><integer : 1></code>	1	the last sequence in the first block of sequences
STOP_ON_ERROR = <code><integer : 1></code>	1	whether to stop on error
OFF_DIAGONAL = <code><integer : 1></code>	100	to speed up the alignment
MATRIX_OFFSET = <code><real : 1></code>	0.00	substitution matrix offset for local alignment
OVERHANG = <code><integer : 1></code>	0	un-penalized overhangs in protein comparisons
LOCAL_ALIGNMENT = <code><logical : 1></code>	off	whether to do local as opposed to global alignment
ALIGN_WHAT = <code><string : 1></code>	'BLOCK'	what to align in ALIGN; 'BLOCK' 'ALIGNMENT' 'LAST'
READ_WEIGHTS = <code><logical : 1></code>	off	whether to read the whole NxM weight matrix for ALIGN*
WRITE_WEIGHTS = <code><logical : 1></code>	off	whether to write the whole NxM weight matrix for ALIGN*
INPUT_WEIGHTS_FILE = <code><string : 1></code>	''	
OUTPUT_WEIGHTS_FILE = <code><string : 1></code>	''	

Output: MODELLER_STATUS = `<integer : 1>`

Description: This command aligns two blocks of sequences.

The two blocks of sequences to be aligned are sequences 1 to ALIGN_BLOCK and ALIGN_BLOCK+1 to the last sequence. The sequences within the two blocks should already be aligned; their alignment does not change.

The command can do either the global (similar to [?]; LOCAL_ALIGNMENT = off) or local dynamic programming alignment (similar to [?]; LOCAL_ALIGNMENT = on).

For the global alignment, set overhang length OVERHANG to more than 0 so that the corresponding number of residues at either of the four termini won't be penalized by any gap penalties (this makes it a pseudo local alignment).

To speed up the calculation, set OFF_DIAGONAL to a number smaller than the shortest sequence length. The alignments matching residues i and j with $|i - j| > \text{OFF_DIAGONAL}$ are not considered at all in the search for the best alignment.

The gap initiation and extension penalties are specified by GAP_PENALTIES_1D. The default values of -900 -50 for the 'as1.sim.mat' similarity matrix were found to be optimal for pairwise alignments of sequences that share from 30% to 45% sequence identity (RS and AŠ, in preparation).

The residue type – residue type scores are read from file RR_FILE. The routine automatically determines whether it has to maximize similarity or minimize distance.

MATRIX_OFFSET applies to local alignment only and influences its length. MATRIX_OFFSET should be somewhere between the lowest and highest residue–residue scores. A smaller value of this parameter will make the local alignments shorter when distance is minimized, and longer when similarity is maximized. This works as follows: The recursively constructed dynamic programming comparison matrix is reset to 0 at position i, j when the current alignment score becomes larger (distance) or smaller (similarity) than MATRIX_OFFSET. Note that this is equivalent to the usual shifting of the residue–residue scoring matrix in the sense that there are two combinations of GAP_PENALTIES_1D and MATRIX_OFFSET values that will give exactly the same alignments irrespective of whether the matrix is actually offset (with 0 used to restart local alignments in dynamic programming) or the matrix is not offset but MATRIX_OFFSET is used as the cutoff

for restarting local alignments in dynamic programming. For the same reason, the matrix offset does not have any effect on the global alignments if the gap extension penalty is also shifted for half of the matrix offset.

The position–position score is an average residue–residue score for all possible pairwise comparisons between the two blocks ($n \times m$ comparisons are done, where n and m are the number of sequences in the two blocks, respectively). The first exception to this is when `ALIGN_WHAT` is set to `'ALIGNMENTS'`, in which case the two alignments defined by `ALIGN_BLOCK` are aligned; *i.e.*, the score is obtained by comparing only equivalent positions between the two alignment blocks (only n comparisons are done, where n is the number of sequences in each of the two blocks). This option is useful in combination with `COMPARE_ALIGNMENTS` and `WRITE_ALIGNMENT` for evaluation of various alignment parameters and methods. The second exception is when `ALIGN_WHAT` is set to `'LAST'`, in which case only the last sequences in the two blocks are used to get the scores. In `'block'`, `'alignment'`, and `'last'` comparisons, penalty for a comparison of a gap with a residue during the calculation of the scoring matrix is obtained from the score file (gap–gap match should have a score of 0.0).

Only the 20 standard residue types, plus `Asx` (changes to `Asn`) and `Glx` (changes to `Gln`) are recognized. Every other unrecognized residue, except for a gap and a chain break, changes to `Gly` for comparison purposes.

If you receive an error message to increase the `MAXRES` constant, you can try to increase the gap penalties first. Here and elsewhere in `MODELLER`, `MAXRES` is both the maximal number of residues in a protein as well as the maximal length of an alignment. If the length of the alignment arrays is too small, `MODELLER_STATUS` becomes 1 (Section 2.1.3).

For the time being, this and the other alignment commands (`MALIGN`, `ALIGN2D`, `ALIGN3D`, and `MALIGN3D`) remove chain break information from the `CALN` array, which means that chain breaks are not retained when the alignment is written to a file after executing these commands.

Example:

```
# Example for: ALIGN

# This will read two sequences, align them, and write the alignment
# to a file:

SET OUTPUT_CONTROL = 1 1 1 1 1

READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = '1fas' '2ctx'
# The as1.sim.mat similarity matrix is used by default:
ALIGN GAP_PENALTIES_1D = -600 -400
WRITE_ALIGNMENT FILE = 'toxin-seq.ali'
```

2.4.15 ALIGN2D — align sequences with structures

Options:

<code>RR_FILE = <string : 1></code>	<code>'\$(LIB)/as1.sim.mat'</code>	input residue-residue scoring file
<code>DIRECTORY = <string : 1></code>	<code>''</code>	directory of <code>RR_FILE</code>
<code>GAP_PENALTIES_1D = <real : 2></code>	<code>-900 -50</code>	gap creation and extension penalties for sequence/sequence alignment
<code>GAP_PENALTIES_2D = <real : 8></code>	<code>0.35 1.2 0.9 1.2 0.6 8.6 1.2 0.0</code>	gap penalties for sequence/structure alignment: helix, beta, accessibility, straightness, and CA–CA distance factor, <code>dst min</code> , <code>dst power</code> , <code>t</code> ; best <code>U,V=-450,0</code>
<code>ALIGN_BLOCK = <integer : 1></code>	<code>1</code>	the last sequence in the first block of sequences
<code>STOP_ON_ERROR = <integer : 1></code>	<code>1</code>	whether to stop on error
<code>OFF_DIAGONAL = <integer : 1></code>	<code>100</code>	to speed up the alignment

MATRIX_OFFSET = <code><real : 1></code>	0.00	substitution matrix offset for local alignment
OVERHANG = <code><integer : 1></code>	0	un-penalized overhangs in protein comparisons
LOCAL_ALIGNMENT = <code><logical : 1></code>	off	whether to do local as opposed to global alignment
FIT = <code><logical : 1></code>	on	whether to align
READ_WEIGHTS = <code><logical : 1></code>	off	whether to read the whole NxM weight matrix for ALIGN*
WRITE_WEIGHTS = <code><logical : 1></code>	off	whether to write the whole NxM weight matrix for ALIGN*
INPUT_WEIGHTS_FILE = <code><string : 1></code>	''	
OUTPUT_WEIGHTS_FILE = <code><string : 1></code>	''	

Output: MODELLER_STATUS = `<integer : 1>`

Description: This command aligns a block of sequences (second block) with a block of structures (first block). It is the same as the **ALIGN** command except that a variable gap **opening** penalty is used. This gap penalty depends on the 3D structure of all sequences in block 1. The variable gap penalty can favor gaps in exposed regions, avoid gaps within secondary structure elements, favor gaps in curved parts of the mainchain, and minimize the distance between the two C_α positions spanning a gap. The **ALIGN2D** command is preferred for aligning a sequence with structure(s) in comparative modeling because it tends to place gaps in a better structural context. See Section 5.1.2 for the dynamic programming algorithm that implements the variable gap penalty. **GAP_PENALTIES_2D** specifies parameters ω_H , ω_S , ω_B , ω_C , ω_D , d_o , γ , and t (Section 5.1.2). The default gap penalties **GAP_PENALTIES_1D** (-450, 0) and **GAP_PENALTIES_2D** (0.35, 1.2, 0.9, 1.2, 0.6, 8.6, 1.2, 0.0) as well as the **RR_FILE** substitution matrix ('as1.sim.mat') were found to be optimal in pairwise alignments of structures and sequences sharing from 30% to 45% sequence identity (RS and AŠ, in preparation).

— move to back

The linear gap penalty function for inserting a gap in block 1 of structures is: $g = f_1(H, S, B, C)u + lv$ where u and v are the usual gap opening and extension penalties, l is gap length, and f_1 is a function that is at least 1, but can be larger to make gap opening more difficult in the following circumstances: between two consecutive (*i.e.*, $i, i + 1$) helical positions, two consecutive β -strand positions, two consecutive buried positions, or two consecutive positions where the mainchain is locally straight. This function is $f_1 = 1 + [\omega_H H_i H_{i+1} + \omega_S S_i S_{i+1} + \omega_B B_i B_{i+1} + \omega_C C_i C_{i+1}]$, H_i is the fraction of helical residues at position i in block 1, S_i is the fraction of β -strand residues at position i in block 1, B_i is the average relative sidechain buriedness of residues at position i in block 1, and C_i is the average straightness of residues at position i in block 1. See Section ?? for the definition of these features. The original straightness is modified here by assigning maximal straightness of 1 to all residues in a helix or a β -strand.

The linear gap penalty function for opening a gap in block 2 of sequences is: $g = f_2(H, S, B, C, D)u + lv$ where f_2 is a function that is at least 1, but can be larger to make the gap opening in block 2 more difficult in the following circumstances: when the first gap position is aligned with a helical residue, a β -strand residue, a buried residue, extended mainchain, or when the whole gap in block 2 is spanned by two residues in block 1 that are far apart in space. This function is $f_2 = 1 + [\omega_H H_i + \omega_S S_i + \omega_B B_i + \omega_C C_i + \omega_D \sqrt{d - d_o}]$. d is the distance between the two C_α atoms spanning the gap, averaged over all structures in block 1 and d_o is the distance that is small enough to correspond to no increase in the opening gap penalty (*e.g.*, 8.6Å).

When **FIT** is off, no alignment is done and the routine returns only the average structural information, which can be written out by the **WRITE_ALIGNMENT** command.

Example:

```
# Demonstrating ALIGN2D, aligning with variable gap penalty

SET OUTPUT_CONTROL = 1 1 1 1 1
```

```

READ_TOPOLOGY FILE = '$(LIB)/top_heav.lib'

# Read aligned structure(s):
READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = '2ctx'
# READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = '2ctx' '2abx'
SET ADD_SEQUENCE = on, ALIGN_BLOCK = NUMB_OF_SEQUENCES
# Read aligned sequence(s):
READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = ALIGN_CODES '1nbt'

# Structure sensitive variable gap penalty sequence-sequence alignment:
SET OVERHANG = 0
# SET RR_FILE = '$(LIB)/id.sim.mat'
SET GAP_PENALTIES_1D = -450 0
SET GAP_PENALTIES_2D = 0.35 1.2 0.9 1.2 0.6 8.6 1.2 0.0
ALIGN2D
WRITE_ALIGNMENT FILE = 'align2d.ali', ALIGNMENT_FORMAT = 'PIR',
WRITE_ALIGNMENT FILE = 'align2d.pap', ALIGNMENT_FORMAT = 'PAP', ;
ALIGNMENT_FEATURES='INDICES HELIX BETA STRAIGHTNESS ACCESSIBILITY CONSERVATION'
CHECK_ALIGNMENT

# Color the first template structure according to gaps in alignment:
READ_ALIGNMENT FILE = 'align2d.ali', ALIGN_CODES = '2ctx' '1nbt', ;
ALIGNMENT_FORMAT = 'PIR', ADD_SEQUENCE = off, REMOVE_GAPS = on
READ_MODEL MODEL_SEGMENT = '2ctx', FILE = '2ctx'
COLOR_ALN_MODEL
WRITE_MODEL FILE = '2ctx.aln.pdb'

# Color the first template structure according to secondary structure:
WRITE_DATA OUTPUT = 'SSM BISO_SSM', FILE = '2ctx'
WRITE_MODEL FILE = '2ctx.ssm.pdb'

# Superpose the target structure onto the first template:
READ_MODEL2 FILE = '1nbt.pdb', MODEL2_SEGMENT = '1nbt' '1nbt'
PICK_ATOMS ATOM_TYPES = 'CA'
SUPERPOSE
WRITE_MODEL2 FILE = '1nbt.fit.pdb'

```

2.4.16 MALIGN — align two or more sequences

Options:

RR_FILE = <string: 1>	'\$(LIB)/as1.sim.mat'	input residue-residue scoring file
DIRECTORY = <string: 1>	''	directory list (e.g., 'dir1:dir2:dir3:./:/')
GAP_PENALTIES_1D = <real: 2>	-900 -50	gap creation and extension penalties for sequence/sequence alignment
OFF_DIAGONAL = <integer: 1>	100	to speed up the alignment
ALIGN_BLOCK = <integer: 1>	1	the last sequence in the first block of sequences
MATRIX_OFFSET = <real: 1>	0.00	substitution matrix offset for local alignment
OVERHANG = <integer: 1>	0	un-penalized overhangs in protein comparisons
LOCAL_ALIGNMENT = <logical: 1>	off	whether to do local as opposed to global alignment
STOP_ON_ERROR = <integer: 1>	1	whether to stop on error

Output: MODELLER_STATUS = \langle integer : 1 \rangle

Description: This command performs a multiple sequence alignment. The sequences to be aligned are the sequences in the current alignment arrays. The command uses the dynamic programming method for the best sequence alignment, given the gap initiation and extension penalties specified by GAP_PENALTIES_1D, and residue type weights read from file RR_FILE. See command **ALIGN** for more information.

The algorithm for the multiple alignment is as follows. First, sequence 2 is aligned with sequence 1 (*i.e.*, block of sequences from 1-ALIGN_BLOCK). Next, sequence 3 is aligned with an average of the aligned sequences 1 and 2; *i.e.*, the weight matrix is an average of the weights 1-3 and 2-3. For this averaging, the gap-residue and gap-gap weights are obtained from the residue-residue weight matrix file, not from gap penalties. If the corresponding weights are not in the file, they are set to the worst and best residue-residue score, respectively. See instructions for **ALIGN** for more details.

Example:

```
# Example for: MALIGN

# This will read all sequences from a file, align them, and write
# the alignment to a new file:

READ_ALIGNMENT_FILE = 'toxin.ali', ALIGN_CODES = 'all'
MALIGN GAP_PENALTIES_1D= -600 -400
WRITE_ALIGNMENT_FILE = 'toxin-seq.pap', ALIGNMENT_FORMAT = 'PAP'
```

2.4.17 ALIGN_CONSENSUS — consensus sequence alignment

Options:

GAP_PENALTIES_1D = \langle real : 2 \rangle	-900 -50	gap creation and extension penalties for sequence/sequence alignment
ALIGN_BLOCK = \langle integer : 1 \rangle	1	the last sequence in the first block of sequences
STOP_ON_ERROR = \langle integer : 1 \rangle	1	whether to stop on error
READ_WEIGHTS = \langle logical : 1 \rangle	off	whether to read the whole NxM weight matrix for ALIGN*
WRITE_WEIGHTS = \langle logical : 1 \rangle	off	whether to write the whole NxM weight matrix for ALIGN*
INPUT_WEIGHTS_FILE = \langle string : 1 \rangle	''	
OUTPUT_WEIGHTS_FILE = \langle string : 1 \rangle	''	

Output: MODELLER_STATUS = \langle integer : 1 \rangle

Description: This command is similar to **ALIGN** except that a consensus alignment of two blocks of sequences is produced. A consensus alignment is obtained from a consensus similarity matrix using the specified gap penalties and the global dynamic programming method. The consensus similarity matrix is obtained by aligning the two blocks of sequences many times with different parameters and methods and counting how many times each pair was aligned. This command is still experimental and no detailed description is given at this time.

This command also produces the alignment accuracy that can be printed out by the **WRITE_ALIGNMENT** command in the 'PAP' format (0 inaccurate, 9 accurate). If the gap initiation penalty is 0, the gap extension penalty of say 0.4 means that only those positions will be equivalenced that were aligned in at least 80% of the individual alignments (*i.e.*, 2 times 0.40).

Example:

```
# Example for: ALIGN_CONSENSUS

# This will read 2 sequences and prepare a consensus alignment
# from many different pairwise alignments.

READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = '2ctx' '2abx'
ALIGN_CONSENSUS GAP_PENALTIES_1D= 0 0.4, ALIGN_BLOCK = 1
WRITE_ALIGNMENT FILE = 'toxin-seq.pap', ALIGNMENT_FORMAT = 'PAP'
```

2.4.18 SUPERPOSE — superpose MODEL2 on MODEL given alignment**Options:**

ALIGN_CODES = \langle string : 0	'all'	codes of proteins in the alignment
FIT = \langle logical : 1	on	whether to superpose
SUPERPOSE_REFINE = \langle logical : 1	off	whether to refine the superposition
RMS_CUTOFFS = \langle real : 11	3.5 3.5 60 60 15 60 60 60 60 60 60	only the first element is used for calculating the cutoff RMS and DRMS measures
REFERENCE_ATOM = \langle string : 1	''	reference atom name in SUPERPOSE
REFERENCE_DISTANCE = \langle real : 1	3.5	cutoff for selecting reference positions in SUPER- POSE
SWAP_ATOMS_IN_RES = \langle string : 1	''	minimize RMS by swapping atoms in these residues (1 char code: 'DEFHLNQRVY')

Requirements: MODEL & MODEL2 [& alignment]

Description: This command superposes MODEL2 on MODEL, without changing the alignment.

If an alignment is in memory, it is used to obtain the equivalent atoms. MODEL must be the first sequence in the alignment, MODEL2 must be the second sequence in the alignment. The equivalent atoms are those selected atoms (set 1) of the MODEL that have equivalently named atoms in MODEL2; the atom equivalences are defined in library \$ATMEQV_LIB. Use the **PICK_ATOMS** command to select the desired atoms for superposition. By default, all atoms are selected. If there is no alignment, a 1:1 correspondence between the residues is assumed.

No fitting is done if FIT = off.

The ALIGN_CODES variable is used only for output, not in calculations.

The RMS_CUTOFFS[1] element is the cutoff used in calculating the cutoff RMS deviations; *i.e.*, those position and distance RMS deviations that are defined on the equivalent atoms which are less than RMS_CUTOFFS[1] angstroms away from each other (as superposed using all aligned positions) and those equivalent distances which are less than RMS_CUTOFFS[1] angstroms different from each other, respectively.

If SUPERPOSE_REFINE is on the refinement of the superposition is done by repeating the fitting with only those aligned pairs of atoms that are within RMS_CUTOFFS[1] of each other until there is no change in the number of equivalent positions. This refinement can only remove compared positions, not add them like **ALIGN3D** can do. This is useful for comparing equivalent parts of two structures with a fixed alignment but omitting divergent parts from the superposition and RMS deviation calculation; *e.g.*, comparing a model with the X-ray structure.

If SUPERPOSE_REFINE is off and REFERENCE_ATOM is non-blank, only those pairs of equivalently named selected atoms from aligned residues are superposed that come from residues whose REFERENCE_ATOM atoms are closer than REFERENCE_DISTANCE Å to each other.

When MODEL and MODEL2 have exactly the same atoms in the same order, one can set SWAP_ATOMS_IN_RES to any combination of single character amino acid residue codes in DEFHLNQRVY. Certain atoms (see below) in the specified sidechains of MODEL2 are then swapped to minimize their RMS deviation relative to MODEL. The labelling resulting in the lowest RMS deviation is retained. The following swaps are attempted:

Residue	Swap(s)
D	OD1, OD2
E	OE1, OE2
F	CD1, CD2 CE1, CE2
H	ND1, CD2 NE2, CE1
N	OD1, ND2
Q	OE1, NE2
R	NH1, NH2
V	CG1, CG2
Y	CD1, CD2 CE1, CE2

Example:

```
# Example for: SUPERPOSE

# This will use a given alignment to superpose Calpha atoms of
# one structure (2ctx) on the other (1fas).

READ_MODEL FILE = '1fas'
READ_MODEL2 FILE = '2ctx'
SET ALIGN_CODES = '1fas' '2ctx'
READ_ALIGNMENT FILE = 'toxin.ali'
PICK_ATOMS PICK_ATOMS_SET = 1, ATOM_TYPES = 'CA'
SUPERPOSE
WRITE_MODEL2 FILE = '2ctx.fit'
```

Example:

```
# Example for: ALIGN3D, SUPERPOSE

# This will align 3D structures of two proteins:

SET OUTPUT_CONTROL = 1 1 1 1 1

# First example: read sequences from a sequence file:
READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = '1fas' '2ctx'
ALIGN GAP_PENALTIES_1D= -600 -400
ALIGN3D GAP_PENALTIES_3D= 0 2.0
WRITE_ALIGNMENT FILE = 'toxin-str.ali'

# Second example: read sequences from PDB files to eliminate the
# need for the toxin.ali sequence file:
READ_MODEL FILE = '1fas'
SEQUENCE_TO_ALI ATOM_FILES = '1fas', ALIGN_CODES = '1fas'
READ_MODEL2 FILE = '2ctx'
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ATOM_FILES = ATOM_FILES '2ctx', ;
```

```

                ALIGN_CODES = ALIGN_CODES '2ctx'
ALIGN GAP_PENALTIES_1D= -600 -400
ALIGN3D GAP_PENALTIES_3D= 0 2.0
WRITE_ALIGNMENT FILE = 'toxin-str.ali'

# And now superpose the two structures using current alignment to get
# various RMS's:
READ_MODEL FILE = '1fas'
PICK_ATOMS ATOM_TYPES = 'CA'
READ_MODEL2 FILE = '2ctx'
SUPERPOSE FIT_ATOMS = 'CA'

```

Example:

```

# This script illustrates the use of the SWAP_ATOMS_IN_RES
# argument to the SUPERPOSE command:

# Need to make sure that the topologies of the two molecules
# superposed are exactly the same:

READ_TOPOLOGY FILE = '$(LIB)/top_heav.lib'
READ_PARAMETERS FILE = '$(LIB)/par.lib'
READ_MODEL FILE = '../tutorial-model/1fdx.atm'
SEQUENCE_TO_ALI ALIGN_CODES = '1fdx', ATOM_FILES = FILE
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ALIGN_CODES = ALIGN_CODES '1fdx', ATOM_FILES = ATOM_FILES FILE
GENERATE_TOPOLOGY SEQUENCE = '1fdx'
TRANSFER_XYZ
BUILD_MODEL INITIALIZE_XYZ = off

# READ_MODEL2 FILE = '../tutorial-model/1fdx.B99990002'
READ_MODEL2 FILE = './1fdx.swap.atm'
SET SWAP_ATOMS_IN_RES = ''
SUPERPOSE
SET SWAP_ATOMS_IN_RES = 'DEFHLNQRVY'
SUPERPOSE FIT = off
SET SWAP_ATOMS_IN_RES = ''
SUPERPOSE FIT = on

```

2.4.19 COMPARE — compare 3D structures given alignment**Options:**

ALIGN_CODES = <string: 0>	'all'	codes of proteins in the alignment
ATOM_FILES = <string: 0>	''	complete or partial atom filenames
ATOM_FILES_DIRECTORY = <string: 1>	'./'	input atom files directory list (e.g., 'dir1:dir2:dir3:./:/')
OUTPUT = <string: 1>	'LONG'	selects output: 'SHORT' 'LONG' 'RMS' 'DRMS'
MATRIX_FILE = <string: 1>	'family.mat'	the filename of the pairwise distance matrix
COMPARE_MODE = <integer: 1>	3	selects the type of comparison: 1 2 3

RMS_CUTOFFS = <code><real : 11></code>	3.5 3.5 60 60 15 60	cutoffs for RMS, DRMS, Alpha Phi Psi Omega
	60 60 60 60 60	chi1 chi2 chi3 chi4 chi5
FIT_ATOMS = <code><string : 1></code>	'CA'	whether to superpose before comparison
DISTANCE_ATOMS = <code><string : 2></code>	'CA' 'CA'	atom type used for variability calculations
FIT = <code><logical : 1></code>	on	whether to do pairwise least-squares fitting or ALIGN2D alignment
ASGL_OUTPUT = <code><logical : 1></code>	off	whether to write output for ASGL

Description: This command compares the structures in the given alignment. It does not make an alignment, but it calculates the RMS and DRMS deviations between atomic positions and distances, and class differences between the mainchain and sidechain dihedral angles. In contrast to the **SUPERPOSE** command, **COMPARE** works with a multiple alignment and it writes more information about the pairwise comparisons.

If no alignment is available, it assumes a 1:1 correspondence for the proteins specified by **ATOM_FILES** or **ALIGN_CODES**. If **ATOM_FILES** is defined, it is used with the MODELLER file-naming mechanism (Section 2.1.4) to find full names for the atom files. If it is not defined, **ALIGN_CODES**, which is usually set by the previous **READ_ALIGNMENT** command, is used. **ALIGN_CODES** does not have to be set if **ATOM_FILES** is set.

OUTPUT selects short ('SHORT') or long ('LONG') form of output to the log file. If it contains word 'RMS' or 'DRMS' it also outputs the RMS or DRMS deviation matrix to file **MATRIX_FILE**. This file can be used with the PHYLIP program or with the **DENDROGRAM** or **PRINCIPAL_COMPONENTS** commands of MODELLER to calculate a clustering of the structures.

COMPARE_MODE selects the form of the positional variability calculated for each position along the sequence:

- 1, for true RMS deviation over all proteins that have a residue at the current position. This does not make any sense for periodic quantities like dihedral angles.
- 2, for the average absolute distance over all pairs of residues that have a residue at the current position.
- 3, the same as 2 except that average distance, not its absolute value is used (convenient for comparison of 2 structures to get the \pm sign of the changes for dihedral angles and distances).

RMS_CUTOFFS specifies cutoff values for calculation of the position, distance, and dihedral angle RMS deviations for pairwise overall comparisons. If difference between two equivalent points is larger than cutoff it is not included in the RMS sum. The order of cutoffs in this vector is: atomic position, intra-molecular distance, α , Φ , Ψ , ω , χ_1 , χ_2 , χ_3 , χ_4 , and χ_5 (there are 5 dihedrals in a disulfide bridge), where α is the virtual C_α dihedral angle between four consecutive C_α atoms. These cutoffs do not affect positional variability calculations.

FIT_ATOMS string specifies all the atom types (including possibly a generic 'ALL') to be fitted in the least-squares superposition. These atom types are used in the least-squares superposition, and in calculation of the position and distance RMS deviations.

DISTANCE_ATOMS[1] specifies the atom type that is used for getting the average structure and RMS deviation at each alignment position in the ASGL output file 'posdif.asgl'. This ASGL file contains the positional variability of the selected atom type in the family of compared proteins. The ASGL output files can then be used with ASGL scripts 'posdif' and 'dih' to produce POSTSCRIPT plots of the corresponding variabilities at each alignment position. **ASGL_OUTPUT** has to be on to obtain the ASGL output files.

If **FIT** = on, a least-squares superposition is done before the comparisons; otherwise, the orientation of the molecules in the input atom files is used.

Example: See **MALIGN3D** command.

2.4.20 ALIGN3D — align two structures

Options:

GAP_PENALTIES_3D = <code><real : 2></code>	0.0 1.75	gap creation and extension penalties for structure/structure superposition
--	----------	--

FIT_ATOMS = \langle string : 1 \rangle	'CA'	one atom type used for superposition
FIT = \langle logical : 1 \rangle	on	whether to align
STOP_ON_ERROR = \langle integer : 1 \rangle	1	whether to stop on error
OUTPUT = \langle string : 1 \rangle	'LONG'	'SHORT' 'LONG' 'VERY_LONG'
ALIGN3D_TRF = \langle logical : 1 \rangle	off	whether to transform the distances before dynamic programming
ALIGN3D_REPEAT = \langle logical : 1 \rangle	off	do several starts to maximize number of equivalent positions
OFF_DIAGONAL = \langle integer : 1 \rangle	100	to speed up the alignment
MATRIX_OFFSET = \langle real : 1 \rangle	0.00	substitution matrix offset for local alignment
OVERHANG = \langle integer : 1 \rangle	0	un-penalized overhangs in protein comparisons
LOCAL_ALIGNMENT = \langle logical : 1 \rangle	off	whether to do local as opposed to global alignment

Output: MODELLER_STATUS = \langle integer : 1 \rangle

Description: This command uses the current alignment as the starting point for an iterative least-squares superposition of two 3D structures. This results in a new pairwise structural alignment. If no alignment is in memory, the initial alignment is the 1:1 alignment. A good initial alignment may be obtained by sequence alignment (**ALIGN**). For superpositions, only one atom per residue is used, as specified by FIT_ATOMS[1].

The alignment algorithm is as follows. First, structure 2 is least-squares fit on structure 1 using all the equivalent residue positions in the initial alignment that have the specified atom type. Next, the residue-residue distance matrix is obtained by calculating Euclidean distances between all pairs of selected atoms from the two structures. The alignment of the two structures is then obtained by the standard dynamic programming optimization based on the residue-residue distance matrix.

GAP_PENALTIES_3D[1] is a gap creation penalty (usually 0), and GAP_PENALTIES_3D[2] is a gap extension penalty, say 1.75. This procedure identifies pairs of residues as equivalent when they have their selected atoms at most 2 times GAP_PENALTIES_3D[2] angstroms apart in the current orientation (this is so when the gap initiation penalty is 0). The reason is that an equivalence costs the distance between the two residues while an alternative, the gap-residue and residue-gap matches, costs twice the gap extension penalty.

From the dynamic programming run, a new alignment is obtained. Thus, structure 2 can be fitted onto structure 1 again, using this new alignment, and the whole cycle is repeated until there is no change in the number of equivalent positions and until the difference in the rotation matrices for the last two superpositions is very small. At the end, the framework, that is the alignment positions without gaps, is written to the log file.

If FIT is off, no alignment is done.

If OUTPUT contains 'SHORT', only the best alignment and its summary are displayed. If OUTPUT contains 'LONG', summaries are displayed for all initial alignments in each framework cycle. If OUTPUT contains 'VERY_LONG', all alignments are displayed.

If ALIGN3D_TRF is on, the weights in the weight matrix are modified distances [?].

If ALIGN3D_REPEAT is on, three additional initial alignments are tried and the one resulting in the largest number of equivalent positions is selected.

Example:

```
# Example for: ALIGN3D, SUPERPOSE

# This will align 3D structures of two proteins:

SET OUTPUT_CONTROL = 1 1 1 1 1

# First example: read sequences from a sequence file:
```

```

READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = '1fas' '2ctx'
ALIGN GAP_PENALTIES_1D= -600 -400
ALIGN3D GAP_PENALTIES_3D= 0 2.0
WRITE_ALIGNMENT FILE = 'toxin-str.ali'

# Second example: read sequences from PDB files to eliminate the
# need for the toxin.ali sequence file:
READ_MODEL FILE = '1fas'
SEQUENCE_TO_ALI ATOM_FILES = '1fas', ALIGN_CODES = '1fas'
READ_MODEL FILE = '2ctx'
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ATOM_FILES = ATOM_FILES '2ctx', ;
                ALIGN_CODES = ALIGN_CODES '2ctx'
ALIGN GAP_PENALTIES_1D= -600 -400
ALIGN3D GAP_PENALTIES_3D= 0 2.0
WRITE_ALIGNMENT FILE = 'toxin-str.ali'

# And now superpose the two structures using current alignment to get
# various RMS's:
READ_MODEL FILE = '1fas'
PICK_ATOMS ATOM_TYPES = 'CA'
READ_MODEL2 FILE = '2ctx'
SUPERPOSE FIT_ATOMS = 'CA'

```

2.4.21 MALIGN3D — align two or more structures

Options:

ALIGN_CODES = <string: 0>	'all'	codes of proteins in the alignment
ATOM_FILES = <string: 0>	''	complete or partial atom filenames
ATOM_FILES_DIRECTORY = <string: 1>	'./'	input atom files directory list (e.g., 'dir1:dir2:dir3:./:/')
GAP_PENALTIES_3D = <real: 2>	0.0 1.75	gap creation and extension penalties for structure/structure superposition
OFF_DIAGONAL = <integer: 1>	100	to speed up the alignment
MATRIX_OFFSET = <real: 1>	0.00	substitution matrix offset for local alignment
OVERHANG = <integer: 1>	0	un-penalized overhangs in protein comparisons
LOCAL_ALIGNMENT = <logical: 1>	off	whether to do local as opposed to global alignment
FIT_ATOMS = <string: 1>	'CA'	one atom type for superposition
FIT = <logical: 1>	on	whether to align
OUTPUT = <string: 1>	'LONG'	'SHORT' 'LONG' 'VERY_LONG' 'NO_ALIGNMENT'
WRITE_FIT = <logical: 1>	off	whether to write out fitted coordinates to .fit files
CURRENT_DIRECTORY = <logical: 1>	on	whether to write output .fit files to current directory
WRITE_WHOLE_PDB = <logical: 1>	on	whether to write out all lines in the input PDB file
STOP_ON_ERROR = <integer: 1>	1	whether to stop on error

Output: MODELLER.STATUS = <integer: 1>

Description: This command uses the current alignment as the starting point for an iterative least-squares superposition of two or more 3D structures. This results in a new multiple structural alignment. If no alignment is in memory, the initial alignment is the 1:1 alignment. A good initial alignment may be obtained by sequence alignment (**MALIGN**). For superpositions, only one atom per residue is used, as specified by **FIT_ATOMS**. The resulting alignment can be written to a file with the **WRITE_ALIGNMENT** command. The multiply superposed coordinates remain in memory and can be used with such commands as **TRANSFER_XYZ** if **ATOM_FILES** is not changed in the meantime. It is best to use the structure that overlaps most with all the other structures as the first protein in the alignment. This may prevent an error exit due to too few equivalent positions during framework construction.

The alignment algorithm is as follows. There are several cycles, each of which consists of an update of a framework and a calculation of a new alignment; the new alignment is based on the superposition of the structures onto the latest framework. The framework in each cycle is obtained as follows. The initial framework consists of the atoms in structure 1 that correspond to **FIT_ATOMS**. If there is no specified atom types in any of the residues at a given position, the coordinates for this framework position are approximated by the neighboring coordinates. Next, all other structures are fit to this framework. The final framework for the current cycle is then obtained as an average of all the structures, in their fitted orientations, but only for residue positions that are common to all of them, given the current alignment. Another result is that all the structures are now superposed on this framework. Note that the alignment has not been changed yet. Next, the multiple alignment itself is re-derived in $N - 1$ dynamic programming runs, where N is the number of structures. This is done as follows. First, structure 2 is aligned with structure 1, using the inter-molecular atom-atom distance matrix, for all atoms of the selected type, as the weight matrix for the dynamic programming run. Next, structure 3 is aligned with an average of structures 1 and 2 using the same dynamic programming technique. Structure 4 is then aligned with an average of structures 1-3, and so on. Averages of structures $i-j$ are calculated for all alignment positions where there is at least one residue in any of the structures $i-j$ (this is different from a framework which requires that residues from all structures be present). Note that in this step, residues out of the current framework may get aligned and the current framework residues may get unaligned. Thus, after the series of $N - 1$ dynamic programming runs, a new multiple alignment is obtained. This is then used in the next cycle to obtain the next framework and the next alignment. The cycles are repeated until there is no change in the number of equivalent positions. This procedure is best viewed as a way to determine the framework regions, not the whole alignment. The results from this command are expected to be similar to the output of program **MNYFIT** [?].

GAP_PENALTIES_3D[1] is a gap creation penalty (usually 0), and **GAP_PENALTIES_3D[2]** is a gap extension penalty, say 1.75. This procedure identifies pairs of positions as equivalent when they have their selected atoms at most 2 times **GAP_PENALTIES_3D[2]** angstroms apart in the current superposition (this is so when the gap initiation penalty is 0), as described for the **ALIGN3D** command.

Argument **OUTPUT** can contain the following values:

- 'SHORT', only the final framework is written to the log file.
- 'LONG', the framework after the alignment stage in each cycle is written to the log file.
- 'VERY_LONG', the framework from the framework stage in each cycle is also written to the log.

If **WRITE_FIT** is on, the fitted atom files are written out in their final fitted orientations. Their filenames are the original filenames with an extension **.fit**.

If **CURRENT_DIRECTORY** is on, the output **.fit** files will go to the current directory. Otherwise, the output will be in the directory with the original files.

If **WRITE_WHOLE_PDB** is on, the whole PDB files are written out; otherwise only the parts corresponding to the aligned sequences are output.

If **FIT** is off, the initial alignment is not changed. This is useful when all the structures have to be superimposed with the initial alignment (**FIT = off** and **WRITE_FIT = on**).

Example:

```
# Example for: MALIGN3D, COMPARE
```

```

# This will read all sequences from a sequence file, multiply align
# their 3D structures, and then also compare them using this alignment.

READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = 'all'
MALIGN GAP_PENALTIES_1D= -600 -400
MALIGN3D GAP_PENALTIES_3D= 0 2.0, WRITE_FIT = on, WRITE_WHOLE_PDB = off
WRITE_ALIGNMENT FILE = 'toxin-str.pap', ALIGNMENT_FORMAT = 'PAP'

# Make two comparisons: no cutoffs, and 3.5A/60 degree cutoffs for RMS, DRMS,
# and dihedral angle comparisons:
COMPARE RMS_CUTOFFS = 999 999 999 999 999 999 999 999 999 999
COMPARE RMS_CUTOFFS = 3.5 3.5 60 60 60 60 60 60 60 60

```

2.4.22 EXPAND_ALIGNMENT — put all models into alignment

Options:

ROOT_NAME = <string:1>	'undf'	root of a filename for filename construction
FILE_ID = <string:1>	'default'	file id for filename construction
EXPAND_CONTROL = <integer:5>	9999 9999 1 10 0	for controlling EXPAND_ALIGNMENT
FILE_EXT = <string:1>	''	file extension for filename construction

Output: alignment

Description: ID1, ID2, ROOT_NAME, FILE_EXT, and FILE_ID are used to construct atom filenames for all the models (Section 2.1.4). Next, all the models are added to the alignment, using the last sequence in the input alignment as the guide. This allows easy multiple superposition of all the templates and models after comparative modeling.

Example:

```

# Example for: EXPAND_ALIGNMENT

# This will add models to the alignment.

READ_ALIGNMENT FILE = 'toxin.ali', ALIGN_CODES = '2ctx' '2abx'
EXPAND_ALIGNMENT EXPAND_CONTROL = 9999 9999 1 3 0, ;
                    ROOT_NAME = '2abx', FILE_ID = '.B', FILE_EXT = ''
WRITE_ALIGNMENT FILE = 'toxin-expand.ali'

```

2.4.23 SEQUENCE_SEARCH — search for similar sequences

Options:

RR_FILE = <string:1>	'\$(LIB)/as1.sim.mat'	input residue-residue scoring file
FILE = <string:1>	'default'	file with the target sequence
ALIGN_CODES = <string:0>	'all'	the code of the target sequence
DIRECTORY = <string:1>	''	directory list (e.g., 'dir1:dir2:dir3:./:/')
GAP_PENALTIES_ID = <real:2>	-900 -50	gap creation and extension penalties for sequence/sequence alignment

OFF_DIAGONAL = \langle integer : 1)	100	to speed up the alignment
MATRIX_OFFSET = \langle real : 1)	0.00	substitution matrix offset for local alignment
OVERHANG = \langle integer : 1)	0	un-penalized overhangs in protein comparisons
LOCAL_ALIGNMENT = \langle logical : 1)	off	whether to do local as opposed to global alignment
SEARCH_CHAINS_LIST = \langle string : 1)	'\$(LIB)/CHAINS_3.0_40	file with sequences
SEARCH_CHAINS_FILE = \langle string : 1)	'\$(LIB)/CHAINS_all.seq'	file with a list of sequence codes
SEARCH_GROUP_LIST = \langle string : 1)	'\$(LIB)/CHAINS_3.0_40	file with 40% groups of sequences
ALIGNMENT_FORMAT = \langle string : 0)	'PIR'	sequence file formats; has to be 'PIR'
ALIGNMENT_FEATURES = \langle string : 0)	'INDICES CONSERVATION'	what alignment features to write out: 'ACCURACY' 'HELIX' 'BETA' 'ACCESSIBILITY' 'STRAIGHTNESS' 'CONSERVATION' 'INDICES' 'ALL' 'GAPS'
REMOVE_GAPS = \langle logical : 1)	on	whether to remove all-gap positions in input alignment
SEARCH_TOP_LIST = \langle integer : 1)	20	the length of the output hits list
OUTPUT = \langle string : 1)	'LONG'	'SHORT' 'LONG'
STOP_ON_ERROR = \langle integer : 1)	1	whether to stop on error
SEARCH_SORT = \langle string : 1)	'LONGER'	which sequence to use for normalization when sorting the hit list: 'SHORTER' 'LONGER'
SEARCH_RANDOMIZATIONS \langle integer : 1)	= 0	number of randomizations for calculating the significance of a sequence/sequence similarity
RAND_SEED = \langle integer : 1)	-8123	random seed from -50000 to -2
FAST_SEARCH = \langle logical : 1)	off	whether to use fast sequence search or not
FAST_SEARCH_CUTOFF = \langle real : 1)	1.0	if FAST_SEARCH is ON only sequences with database scan significance higher than this value are considered for randomization significance
DATA_FILE = \langle logical : 1)	off	whether results go to a separate file or not
SIGNIF_CUTOFF = \langle real : 2)	4.0 5.0	cutoff for adding sequences to alignment, max difference from the best

Output: MODELLER_STATUS = \langle integer : 1)

Description: This command searches a sequence database for proteins that are similar to a given target sequence.

Target sequence is read from file FILE.

ALIGN_CODES specifies the code of the target sequence in the FILE file. If only one sequence is in the file, you can use ALIGN_CODES = 'all' to read it without bothering about the actual sequence code.

SEARCH_CHAINS_LIST specifies a file that contains protein codes for the proteins to be compared with the target sequence.

The database sequences specified in SEARCH_CHAINS_LIST file must occur in the SEARCH_CHAINS_FILE file.

The command uses the dynamic programming method for the best sequence alignment, given the gap creation and extension penalties specified by GAP_PENALTIES_1D and residue type scores read from file RR_FILE. GAP_PENALTIES_1D[1] is a gap creation penalty and GAP_PENALTIES_1D[2] is a gap extension penalty.

The SEARCH_TOP_LIST top hits are written to the log file at the end. The hits are sorted according to the fractional sequence identity score obtained by dividing the number of identical residue pairs by the length of the longer sequence (SEARCH_SORT = 'LONGER') or the shorter sequence (SEARCH_SORT = 'SHORTER').

The final list of hits contains three different significance values:

1. SIGNI. Z-score from sequence randomizations. This is the most accurate significance score, but the slowest one to calculate. For each pairwise comparison, the two sequences are shuffled a specified number of times (SEARCH_RANDOMIZATIONS) to obtain the mean and standard deviation of "random" scores from which the Z-score for an alignment score of a given pair of sequences is calculated.

2. **SIGNI2**. Z-score for sequence identity from the database scan. After comparison of the target sequence with all sequences in the database is done, the comparisons are sorted by the length of the database sequence. The pairwise sequence identities of the 20 sequences closest in length to the target sequence are used to calculate the average and standard deviation of the percentage sequence identities for subsequent calculation of the Z-score for the percentage sequence identity of a given pairwise alignment.
3. **SIGNI3**. Z-score for alignment score from the database scan. The procedure is the same as for **SIGNI2**, except that the alignment scores are used instead of the pairwise sequence identities.

The calculation of the Z-scores assumes that the random scores are distributed according to the Gaussian distribution, instead of the extreme value distribution [?], which is more correct.

SEARCH_RANDOMIZATIONS specifies how many alignments of the shuffled sequences are done to calculate the significance score for the overall sequence similarity. If 0, the significance is not calculated. If more than 5 randomizations are done, the significance score, not sequence identity, is used for sorting the hit list.

When **FAST_SEARCH** is on only those sequences that have a database-scan alignment score significance (**SIGNI3** in output) above **FAST_SEARCH_CUTOFF** are used for the “full” randomization-based significance calculation. Since the mean and the standard deviation of the distribution obtained by randomizing the two compared sequences are much more appropriate than the corresponding quantities for the target/database comparisons, **FAST_SEARCH** should be on only when you are in a hurry and the database is large.

If **DATA_FILE** is on the final results (list of PDB codes with significances, *etc.*) are also written to a separate file 'seqsearch.dat'.

If **OUTPUT** is 'LONG', the best alignment for each sequence in **SEARCH_CHAINS_FILE** and its various scores are also written to the log file. If **OUTPUT** is 'VERY_LONG', individual scores obtained for randomized sequences are also written to the log file (this is almost never needed).

If the selected significance score is larger than **SIGNIF_CUTOFF[1]** and not more than **SIGNIF_CUTOFF[2]** units worse than the best hit, all the members of the same group, as defined in **SEARCH_GROUP_LIST**, are added to the alignment array. Subsequent **MALIGN**, **DENDROGRAM** and **WRITE_ALIGNMENT** can then be used to write out all related PDB chains aligned to the target sequence.

Example:

```
# Example for: SEQUENCE_SEARCH

# This will search the MODELLER database of representative protein chains
# for chains similar to the specified sequence.

SET OUTPUT_CONTROL = 1 1 1 1 2
SET SEARCH_RANDOMIZATIONS = 20 # should use 100 in real life;
SET GAP_PENALTIES_1D = -800 -400
SET SEARCH_CHAINS_LIST = 'very-short-for-test.cod'
SEQUENCE_SEARCH FILE = 'toxin.ali', ALIGN_CODES = '1nbt'
MALIGN
WRITE_ALIGNMENT FILE = 'toxin-search.pap', ALIGNMENT_FORMAT = 'PAP'

SET SEARCH_CHAINS_LIST = 'very-short-for-test.cod'
SEQUENCE_SEARCH FILE = 'toxin.ali', ALIGN_CODES = '1nbt'
```

2.4.24 DELETE_ALIGNMENT — delete alignment

Description: This command deletes an existing alignment from the MODELLER memory. This is useful when a default 1:1 correspondence, such as that between an X-ray structure and its MODELLER model, is needed. This default alignment is constructed for the commands that need an alignment only if there is no alignment already in memory.

Example: See **PATCH** command.

2.4.25 SEGMENT_MATCHING — align segments

Options:

RR_FILE = \langle string : 1 \rangle	'\$(LIB)/as1.sim.mat'	input residue-residue scoring file
ALIGN_BLOCK = \langle integer : 1 \rangle	1	the last sequence in the first block of sequences
SEGMENT_REPORT = \langle integer : 1 \rangle	1D6	for SEGMENT_MATCHING
SEGMENT_CUTOFF = \langle real : 1 \rangle	999999	cutoff for writing out an alignment in SEGMENT_MATCHING
SEGMENT_SHIFTS = \langle integer : 0 \rangle		segment shifts +- in SEGMENT_MATCHING
SEGMENT_GROWTH_N = \langle integer : 0 \rangle		reducing/growing segment N-termini in SEGMENT_MATCHING
SEGMENT_GROWTH_C = \langle integer : 0 \rangle		reducing/growing segment C-termini in SEGMENT_MATCHING
MIN_LOOP_LENGTH = \langle integer : 0 \rangle		inter-segment minimal lengths in SEGMENT_MATCHING
FILE = \langle string : 1 \rangle	'default'	partial or complete filename
OUTPUT_DIRECTORY = \langle string : 1 \rangle	''	output directory
ROOT_NAME = \langle string : 1 \rangle	'undf'	root of a filename for filename construction
FILE_ID = \langle string : 1 \rangle	'default'	file id for filename construction
FILE_EXT = \langle string : 1 \rangle	''	file extension for filename construction

Requirements: alignment

Description: This command enumerates alignments between two blocks of sequences. More precisely, it enumerates the alignments between the segments in the first block and the sequences in the second block. The segments can be moved to the left and right as well as lengthened and shortened, relative to the initial alignment. The regions not in segments or not aligned with segments are left un-aligned, possibly to be modeled as insertions. Typically, the first block of sequences corresponds to structures, the segments to secondary structure elements in the first block, and the second block to the sequences one of which is to be modeled later on. The command is useful for generating many alignments which can then be used by another MODELLER script to generate and evaluate the corresponding 3D models.

All the sequences and segments are defined in the alignment array. The first block of sequences, the ones with segments, are the first ALIGN_BLOCK sequences. The regions corresponding to the segments are defined by the last entry in the alignment as contiguous blocks of non-gap residues. Any standard single character residue code may be used. The segments must be separated by gap residues, '-'. The remaining sequences from ALIGN_BLOCK + 1 to NSEQ - 1 are the second block of sequences. The alignment of the sequences within the two blocks does not change. A sample alignment file is

The enumeration of alignments explores all possible combinations of alignments between each segment and the 2nd block of sequences: The starting position of each segment i is varied relative to the input alignment in the interval from SEGMENT_SHIFT[2*i* - 1] to SEGMENT_SHIFT[2*i*]. There has to be at least MIN_LOOP_LENGTH[*i*] and MIN_LOOP_LENGTH[*i* + 1] residues that are not in any segment before and after the i -th segment, respectively. The location of the N-terminus of segment i is varied relative to the location in the input alignment in the interval from SEGMENT_GROWTH_N[2*i* - 1] to SEGMENT_GROWTH_N[2*i*]. Similarly, the location of the C-terminus of segment i is varied relative to the location in the input alignment in the interval from SEGMENT_GROWTH_C[2*i* - 1] to SEGMENT_GROWTH_C[2*i*]. The shortening and lengthening of the segments may be useful in determining the best anchor regions for modeling of a loop.

Each alignment is scored according to the similarity scoring matrix specified by filename RR_FILE. This matrix may contain residue—gap scores, the gap being residue type 21; otherwise the value is set to the smallest value in the matrix. The score for an alignment is obtained by summing scores only over all alignment positions corresponding to the segments (no gap penalty is added for loops). When there is more than one sequence in any of the two blocks, the position score is an average of all pairwise comparisons between the two blocks of sequences. In the case where the number of positions in the alignment changes (*i.e.*, the segments grow

or shorten), the scores are not comparable to each other. It is feasible to enumerate on the order of 10^{10} different alignments in less than one hour of CPU time.

In general, two runs are required. In the first run, the alignments are scored and a histogram of the scores is written to file `FILE`. Then this file must be inspected to determine the cutoff `SEGMENT_CUTOFF`. In the second run, all the alignments with a score higher than `SEGMENT_CUTOFF` are written to files in the PIR format, using the standard filenaming convention: `OUTPUT_DIRECTORY/ROOT_NAMEFILE_IDnnnn0000FILE_EXT`, where `nnnn` is the alignment file counter. In addition, the alignments are also written out in the PAP format for easier inspection by eye. Thus, `SEGMENT_CUTOFF` has to be set to a very large value in the first run, to avoid writing alignment files. During a run, a message is written to the `log` every `SEGMENT_REPORT` alignments; this is useful for knowing what is going on during very long runs.

2.5 Calculation of spatial restraints

This Chapter explains how the restraints are represented in a restraint file and also describes commands for reading, writing, generating, and manipulating restraints. See Section 5.3 for equations defining the restraints and their derivatives with respect to atomic positions. See Section 2.6 for commands for calculating the objective function and Section 5.2 for optimization methods. See the original papers for the most detailed definition and description of the restraints [?, ?].

2.5.1 Specification of restraints

Static and dynamic restraints

Static restraints are read from the restraints file or are generated by the MAKE_RESTRAINTS command. All other restraints are *dynamic* restraints and are created on the fly; they currently include restraints on non-bonded atom pairs.

Formats of the restraints file

Restraints may be read from a restraints file in two formats, MODELLER or USER. The files in the MODELLER and USER formats have to begin with the lines 'MODELLER12 VERSION: MODELLER FORMAT' and 'MODELLER12 VERSION: USER FORMAT', respectively. In both formats, there is one entry per line. The format is free, except that the first character has to be at the beginning of the line. There are three different entry types in the MODELLER format:

```
R Form Modality Feature Group Numb_atoms Numb_parameters 0 Atom_indices Parameters
E Atom_index_1 Atom_index_2
P Pseudo_atom_index Pseudo_atom_type Numb_real_atoms Real_atom_indices
```

For example,

```
R 3 1 1 1 2 2 0 437 28 1.5000 0.1000
E 120 540
P 1 3 3 120 121 122
```

When the line starts with 'R', it contains a restraint, 'E' indicates a pair of atoms to be excluded from the calculation of the dynamic non-bonded pairs list, and 'P' indicates a pseudo atom definition (Section 2.5.2).

The USER format recognizes only the R entries. The fields of a line in the USER format are:

```
Id Form Modality Feature Group Numb_atoms Numb_parameters 0 Parameters Atom_ids
```

For example,

```
R 3 1 1 1 2 2 0 1.5000 0.1000 NH#:1:A CA:2:A
```

The seven integer indices used to specify various restraint properties are listed in Tables 2.2–2.4. They are: **Form** specifies the mathematical form of the restraint. **Modality** should be viewed as the argument to **Form**. It specifies the number of single Gaussians in a poly-Gaussian pdf, periodicity n of the cosine in the cosine potential, and the number of spline points for cubic splines. Only certain combinations of **Form** and **Modality** are possible. Any **Feature** can be used with any **Form/Modality** pair. **Group** or “physical feature type” groups restraints for reporting purposes in **ENERGY**, *etc.* The number of atoms and parameters for the restraint are specified by **Numb_atoms** and **Numb_prms**, respectively. The seventh integer index can be ignored. **Atom_indices** and **Parameters** have to match the hard-wired conventions. The format of the atom id is ATOM_NAME:RESIDUE_#[CHAIN_ID], where ATOM_NAME is the four character IUPAC atom name as found in a PDB file, RESIDUE_# is a five character residue number as it occurs in the PDB file of a model, and the optional CHAIN_ID is the single character chain id as it occurs in the PDB file. For example, the carbonyl oxygen (O) in residue '10A' in chain 'A' is specified by 'O:10A:A'; if the chain has no chain id, the name would be only 'O:10A'.

2.5.2 Specification of pseudo atoms

There are virtual and pseudo atoms. A virtual atom is an atom that occurs in the actual molecule, but whose position is not represented explicitly in the MODEL and topology file. A pseudo atom is a position that does not correspond to an actual atom in a molecule, but is some sort of an average of positions of real atoms. MODELLER follows GROMOS definitions for the seven types of pseudo and virtual atoms: gravity center, V41, V31, P2, V42, P3, and P6. These names are constructed using the following rules: 'V' and 'P' indicate virtual and pseudo atoms, respectively. The second digit indicates the number of substituents on the central atom (for 'V') and the number of protons whose positions are averaged (for 'P'). The last digit indicates the number of protons on the central atom (for 'V').

```
GROMOS ROUTINE #DEF DESCRIPTION
TYPE NAME ATM
```

1	PSD	N	gravity center
2	VCH1	4	virtual aliphatic proton on a tetrahedral carbon (->CH), defined by the central C and the three other substituents;
3	VCH1A	3	virtual aromatic proton on a trigonal carbon (=CH), defined by the central C and the two C atoms bonded to the central C;
4	PCH2	3	pseudo aliphatic proton on a tetrahedral carbon (>CH2) not assigned stereospecifically; its position is between the two real protons; defined by the central C and the other two substituents;
5	VCH2	3	virtual aliphatic proton on a tetrahedral carbon (>CH2) assigned stereospecifically; defined by the central tetrahedral atom and the other two substituents on it;
6	PCH31	2	pseudo aliphatic proton on a tetrahedral carbon (-CH3), defined by the central C and the heavy atom X in X-CH3; its position is the average of the three real protons;
7	PCH32	3	pseudo aliphatic proton between two unassigned -CH3 groups; defined by X in CH3 - X - CH3 and the two C atoms from the two CH3 groups (Val, Leu!); its position is the average of the six real protons;
0	-	-	delta and epsilon protons on rapidly flipping aromatic rings should refer directly to real gamma and delta C atoms, respectively.

In a restraints file, pseudo atoms are indexed from NATM+1 to NATM+NPSEUDO where NPSEUDO is the number of pseudo atoms. The restraints (the R entries) are exactly the same as for the real atoms, except that the pseudo atom integer indices are used (indices are larger than NATM). The pseudo atoms are defined in the P entries:

```
P i j k a1 a2 a3
```

where *i* is atom index of pseudo atom *i*, *j* is the type of the pseudo atom *i* (see the table above), *k* is the number of real atoms defining the current pseudo atom (3 in this case), and *a1 a2 a3* are the integer indices of real atoms defining the current pseudo atom.

For example, if you want to define a pseudo atom which is a gravity center of atoms 4, 7, and 10, and there are 101 real atoms in the protein:

P 102 1 3 4 7 10

#	Form	Parameters	Violation	Reference
1	left Gaussian (harmonic lower bound)	f, σ	$(f - f)/\sigma$	Eq. 5.54
2	right Gaussian (harmonic upper bound)	f, σ	$(f - f)/\sigma$	Eq. 5.55
3	single Gaussian (harmonic potential)	\bar{f}, σ	$(f - \bar{f})/\sigma$	Eq. 5.39
4	multiple Gaussian	$(\omega_i)_{n_i} (\bar{f}_i)_{n_i} (\sigma_i)_n$	$\max_{\omega_i} (f - \bar{f}_i)/\sigma_i$	Eq. 5.50
5	Lennard-Jones potential	A, B	0.0	Eq. 5.62
6	Coulomb point-to-point potential	q_1, q_2	0.0	Eq. 5.59
7	Cosine potential	a, b	c	Eq. 5.56
8	undefined			
9	multiple binormal	$(\omega_i)_{n_i} (\bar{f}_i)_{n_i} (\bar{f}_2i)_{n_i} (\sigma_{1i}, \sigma_{2i})_n, (p)_n$	$\max_{\omega_i} \sqrt{\frac{1}{-2(1-\sigma_i^2)}} \left[\left(\frac{L_1 - \bar{f}_{1i}}{\sigma_{1i}} \right)^2 - 2p_i \frac{L_1 - \bar{f}_{1i}}{\sigma_{1i}} \frac{L_2 - \bar{f}_{2i}}{\sigma_{2i}} + \left(\frac{L_2 - \bar{f}_{2i}}{\sigma_{2i}} \right)^2 \right]$	Eq. 5.50
10	cubic spline	$p_i, \text{ for } i = 1, 6 + n$	$(f - f_{min})/\sigma$	Eq. 5.69

Table 2.2: *List of mathematical forms of restraints.* The parameters and their order in the restraint file are also given (Params above). $(\dots)_n$ indicates that (\dots) is repeated n times, where n is specified by the second integer parameter of the restraint, `modality` (see above). `Modality` also defines periodicity of the cosine restraint, corresponding to parameter n in Eq 5.56, and the number of interpolating points for the spline restraint (Eq. 5.69). Feature f can generally be either a measure of solvent exposure (undocumented), a distance, an angle, or a dihedral angle, with the exception of restraint form 9 that only works with a pair of dihedral angles. The angle unit in the restraints file is radians. The internal angle unit of MODELLER is radians, too. Column ‘Violation’ defines the “relative heavy violations” used in **PICK_HOT_ATOMS**. For cubic splines, f_{min} is the feature value that results in the smallest value of the restraint and σ is the standard deviation of the Gaussian function fitted locally around f_{min} . The parameters p_i for a spline restraint are: the scaling factor (p_1), the smallest value at which interpolation is done, x_1 (p_2), the largest interpolating value x_n (p_3), the interval between interpolating points, Δx (p_4), the first derivative at x_1 (p_5), the first derivative at x_n (p_6). The following n values are the values of the restraint at the interpolating x_i points. The MODELLER-4 format has additional n values, which are the second derivatives of the restraint at the interpolating x_i points.

Index	Feature
1	distance
2	angle
3	dihedral angle
4	a pair of dihedral angles (points 1–4 and 5–8)
5	distance between gravity centers of two groups of atoms
6	minimal distance between several pairs of atoms
7	atomic area exposed to solvent in \AA^2
8	number of neighbouring atoms

Table 2.3: *List of feature types that can be restrained.*

Index	Group
1	Bond length potential
2	Bond angle potential
3	Stereochemical cosine dihedral potential
4	Stereochemical improper dihedral potential
5	soft-sphere overlap restraints
6	Lennard-Jones 6–12 potential
7	Coulomb point-point electrostatic potential
8	H-bonding potential
9	Distance restraints 1 (C_α – C_α)
10	Distance restraints 2 (N–O)
11	Mainchain Φ dihedral restraints
12	Mainchain Ψ dihedral restraints
13	Mainchain ω dihedral restraints
14	Sidechain χ_1 dihedral restraints
15	Sidechain χ_2 dihedral restraints
16	Sidechain χ_3 dihedral restraints
17	Sidechain χ_4 dihedral restraints
18	Disulfide distance restraints
19	Disulfide angle restraints
20	Disulfide dihedral angle restraints
21	X lower bound distance restraints
22	X upper bound distance restraints
23	Distance restraints 3 (SDCH–MNCH)
24	Sidechain χ_5 dihedral restraints
25	(Φ , Ψ) binomial dihedral restraints
26	Distance restraints 4 (SDCH–SDCH)
27	Distance restraints 5 (X–Y)
28	NMR distance restraints 6 (X–Y)
29	NMR distance restraints 7 (X–Y)
30	Minimal distance restraints
31	Non-bonded spline restraints
32	Atomic accessibility restraints
33	Atom density restraints

Table 2.4: *List of “physical” restraint types.*

2.5.3 MAKE_RESTRAINTS — make restraints

Options:

RESTRAINT_TYPE = <code><string : 0></code>	'STEREO'	restraint type to be calculated: 'STEREO' 'BOND' 'ANGLE' 'IMPROPER' 'DIHEDRAL' 'MRFP_STEREO' 'MRFP_BOND' 'MRFP_ANGLE' 'MRFP_DIHEDRAL' 'SPHERE' 'SPHERE14' 'LJ' 'LJ14' 'COULOMB' 'COULOMB14' 'ALPHA' 'STRAND' 'SHEET' 'DISTANCE' 'USER_DISTANCE' 'NONB_PAIR_SPLINE' 'PHI-PSI_BINORMAL' 'PHI-PSI_CLASS' 'PHI_DIHEDRAL' 'PSI_DIHEDRAL' 'OMEGA_DIHEDRAL' 'CHI1_DIHEDRAL' 'CHI2_DIHEDRAL' 'CHI3_DIHEDRAL' 'CHI4_DIHEDRAL'
RADII_FACTOR = <code><real : 1></code>	0.82	factor for van der Waals radii
TOPOLOGY_MODEL = <code><integer : 1></code>	3	selects topology library: 1-9
DIH_LIB_ONLY = <code><logical : 1></code>	off	whether to use only library, not homologs for dihedral angle rsrs
MNCH_LIB = <code><integer : 1></code>	1	which MNCH lib to use in MAKE_RESTRAINTS
INTERSEGMENT = <code><logical : 1></code>	on	whether to restrain inter-segment non-bonded pairs
ADD_RESTRAINTS = <code><logical : 1></code>	off	whether to add new restraints to existing restraints
RESIDUE_GROUPING = <code><integer : 1></code>	1	
MAXIMAL_DISTANCE = <code><real : 1></code>	999.	maximal distance for distance restraints
RESIDUE_SPAN_RANGE = <code><integer : 2></code>	0 99999	range of residues spanning the allowed distances; for MAKE_RESTRAINTS, PICK_RESTRAINTS, non-bonded dynamic pairs
RESIDUE_SPAN_SIGN = <code><logical : 1></code>	on	whether to do $N*(N-1)/2$ loop for atom pairs in MAKE_RESTRAINTS RESTRAINT_TYPE = 'distance'
RESTRAINT_SEL_ATOMS = <code><integer : 1></code>	1	a restraint other than non-bonded pair has to have at least as many selected atoms
NONBONDED_SEL_ATOMS = <code><integer : 1></code>	1	a non-bonded pair has to have at least as many selected atoms
EXCL_LOCAL = <code><logical : 4></code>	on on on on	whether to exclude bonds, angles, dihedrals, explicit excl pairs from the homology-derived distance rsrs
ACCESSIBILITY_TYPE = <code><integer : 1></code>	8	type of solvent accessibility: 1-10
DISTANCE_RSR_MODEL = <code><integer : 1></code>	1	the model for calculating distance restraints: 1-7
RESTRAINT_STDEV = <code><real : 2></code>	0.0 1.0	transforming factors for standard deviations ($y=a+bx$) in models 1-6 or standard deviation for model 7 (a)
RESTRAINT_PARAMETERS = <code><real : 0></code>	3 1 3 3 4 2 0 0.0 0.087	restraint parameters for 'USER_DISTANCE'
MDT_LIB_FILE = TYPEVALUES	DEFAULT	DESCRIPTION
BIN_LIB_FILE = TYPEVALUES	DEFAULT	DESCRIPTION
ATOM_FILES_DIRECTORY = <code><string : 1></code>	'./'	input atom files directory list (e.g., 'dir1:dir2:dir3:./:/')
BASIS_PDF_WEIGHT = <code><string : 1></code>	'LOCAL'	a method for calculation of basis pdf weights: 'LOCAL' 'GLOBAL'
BASIS_RELATIVE_WEIGHT = <code><real : 1></code>	0.05	the cutoff weight of basis pdf's for their removal

RESIDUE_IDS = <code><string : 0></code>	''	residue id (number:chnid)
SPLINE_ON_SITE = <code><logical : 1></code>	off	whether to convert restraints to splines

Requirements: topology & parameters [& alignment] [& picked atoms sets 2 and 3]

Description: This command calculates and selects new restraints of a specified type. See the original papers for the most detailed definition and description of the restraints [?, ?]. The calculation of restraints of all types is now (partly) limited to the selected atoms only (either set 1, or 2 and 3; see below).

If ADD_RESTRAINTS is off, all old restraints are deleted, otherwise new restraints are added to the old ones.

RESTRAINT_TYPE selects the types of the generated restraints. Only one restraint type can be selected at a time, except for the stereochemical restraints (BOND, ANGLE, DIHEDRAL, IMPROPER) that can all be calculated at the same time. It is useful to distinguish between the stereochemical restraints and homology-derived restraints. The stereochemical restraints are obtained from libraries that depend on atom and/or residue types only (e.g., CHARMM 22 force field [?] or statistical potentials), and do not require an alignment with template structures. In contrast, the homology-derived restraints are calculated from related protein structures, which correspond to all but the last sequence in the alignment (the target). These templates are read from coordinate files, which are the only data files required. All restraints are added to the existing restraints, even if they duplicate them (but see the comment for the 'OMEGA' restraints below).

- • **Stereochemical restraints:**

- 'BOND'. This calculates covalent bond restraints (harmonic terms). It relies on the list of the atom-atom bonds for MODEL, prepared previously by the **GENERATE_TOPOLOGY** command. The mean values and force constants are obtained from the parameter library in memory. Only those bonds are restrained that have all or at least RESTRAINT_SEL_ATOMS in the selected atom set 1.
- 'ANGLE'. This calculates covalent angle restraints (harmonic terms). It relies on the list of the atom-atom-atom bonds for MODEL, prepared previously by the **GENERATE_TOPOLOGY** command. The mean values and force constants are obtained from the parameter library in memory. Only those angles are restrained that have all or at least RESTRAINT_SEL_ATOMS in the selected atom set 1.
- 'DIHEDRAL'. This calculates covalent dihedral angle restraints (cosine terms). It relies on the list of the atom-atom-atom-atom dihedral angles for MODEL, prepared previously by the **GENERATE_TOPOLOGY** command. The minima, phases, and force constants are obtained from the parameter library in memory. Only those dihedral angles are restrained that have all or at least RESTRAINT_SEL_ATOMS in the selected atom set 1.
- 'IMPROPER'. This calculates improper dihedral angle restraints (harmonic terms). It relies on the list of the improper dihedral angles for MODEL, prepared previously by the **GENERATE_TOPOLOGY** command. The mean values and force constants are obtained from the parameter library in memory. Only those improper angles are restrained that have all or at least RESTRAINT_SEL_ATOMS in the selected atom set 1.
- 'STEREO'. This implies all 'BOND', 'ANGLE', 'DIHEDRAL', and 'IMPROPER' restraints.
- 'MRFP_BOND'. Similar to 'BOND' except that spline restraints from the corresponding MRFP entries in the parameter library are used instead of the harmonic terms. Only those bonds are restrained that have all or at least RESTRAINT_SEL_ATOMS in the selected atom set 1.
- 'MRFP_ANGLE'. Similar to 'ANGLE' except that spline restraints from the corresponding MRFP entries in the parameter library are used instead of the harmonic terms. Only those angles are restrained that have all or at least RESTRAINT_SEL_ATOMS in the selected atom set 1.
- 'MRFP_DIHEDRAL'. Similar to 'DIHEDRAL' except that spline restraints from the corresponding MRFP entries in the parameter library are used instead of the cosine terms. Only those dihedral angles are restrained that have all or at least RESTRAINT_SEL_ATOMS in the selected atom set 1.
- 'MRFP_STEREO'. This implies all 'MRFP_BOND', 'MRFP_ANGLE', and 'MRFP_DIHEDRAL' restraints.

- 'SPHERE14'. This constructs soft-sphere overlap restraints (lower harmonic bounds) for atom pairs separated by exactly three bonds (1-4 pairs). It relies on atom radii from the '\$RADII14_LIB' library. Only those non-bonded pairs are restrained that have all or at least NONBONDED_SEL_ATOMS in the selected atom set 1. They must also satisfy the RESIDUE_SPAN_RANGE & RESIDUE_SPAN_SIGN criterion.
- 'LJ14'. This constructs 1-4 Lennard-Jones restraints using the modified 1-4 Lennard-Jones parameters from the CHARMM parameter library. There is no way to calculate 'LJ14' as dynamic restraints. Only those non-bonded pairs are restrained that have all or at least NONBONDED_SEL_ATOMS in the selected atom set 1. They must also satisfy the RESIDUE_SPAN_RANGE & RESIDUE_SPAN_SIGN criterion.
- 'COULOMB14'. This constructs 1-4 Coulomb restraints by relying on the atomic charges from the CHARMM topology library. There is no way to calculate 'COULOMB14' as dynamic restraints. Only those non-bonded pairs are restrained that have all or at least NONBONDED_SEL_ATOMS in the selected atom set 1. They must also satisfy the RESIDUE_SPAN_RANGE & RESIDUE_SPAN_SIGN criterion.
- 'SPHERE'. This constructs soft-sphere overlap restraints (lower harmonic bounds) for all atom pairs that are not in bonds, angles, dihedral angles, improper dihedral angles, nor are explicitly excluded by the 'E' entries read from a restraint file or added by the **ADD_RESTRAINT** command. Only those non-bonded pairs are restrained that have all or at least NONBONDED_SEL_ATOMS in the selected atom set 1. They must also satisfy the RESIDUE_SPAN_RANGE & RESIDUE_SPAN_SIGN criterion. Note that this makes these restraints static (*i.e.*, not dynamic) and that you must set DYNAMIC_SPHERE to **off** before evaluating the molecular pdf if you want to avoid duplicated restraints. These restraints should usually not be combined with the Lennard-Jones ('LJ') restraints.

When INTERSEGMENT is on, the inter-segment non-bonded restraints are also constructed; otherwise, the segments do not feel each other *via* the non-bonded restraints. This option does not apply to the **OPTIMIZE** command where information about segments is not used at all (*i.e.*, **OPTIMIZE** behaves as if INTERSEGMENT = on).

- 'LJ'. This constructs Lennard-Jones restraints for all atom pairs that are not in bonds, angles, dihedral angles, improper dihedral angles, nor are explicitly excluded by the 'E' entries read from a restraint file or added by the **ADD_RESTRAINT** command. Only those non-bonded pairs are restrained that have all or at least NONBONDED_SEL_ATOMS in the selected atom set 1. They must also satisfy the RESIDUE_SPAN_RANGE & RESIDUE_SPAN_SIGN criterion. Note that this command makes the non-bonded restraints static (*i.e.*, not dynamic) and that you must set DYNAMIC_LENNARD to **off** before evaluating the molecular pdf if you want to avoid duplicated restraints. Note that CHARMM uses both 'LJ14' and 'LJ'. For large molecules, it is better to calculate 'LJ' as dynamic restraints because you can use distance cutoff CONTACT_SHELL in **OPTIMIZE** to reduce significantly the number of non-bonded atom pairs.
- 'COULOMB'. This constructs Coulomb restraints for all atom pairs that are not in bonds, angles, dihedral angles, improper dihedral angles, nor are explicitly excluded by the 'E' entries read from a restraint file or added by the **ADD_RESTRAINT** command. Only those non-bonded pairs are restrained that have all or at least NONBONDED_SEL_ATOMS in the selected atom set 1. They must also satisfy the RESIDUE_SPAN_RANGE & RESIDUE_SPAN_SIGN criterion. Note that this command makes the non-bonded restraints static (*i.e.*, not dynamic) and that you must set DYNAMIC_COULOMB to **off** before evaluating the molecular pdf if you want to avoid duplicated restraints. Note that CHARMM uses both 'COULOMB14' and 'COULOMB'. For large molecules, it is better to calculate 'COULOMB' as dynamic restraints because you can use distance cutoff CONTACT_SHELL in **OPTIMIZE** to reduce significantly the number of non-bonded atom pairs.
- 'ALPHA'. This makes restraints enforcing an α -helix (mainchain conformation class "A") for the residue segment specified by the two RESIDUE_IDS (Section 2.4.1). The helix is restrained by Φ, Ψ binormal restraints, N-O hydrogen bonds, $C_\alpha-C_\alpha$ distances for $i-j \in \{2-9\}$, $C_\alpha-O$ distances for $i-j \in \{2-9\}$, and O-O distances for $i-j \in \{2-6\}$. These target distances were all obtained from a regular α -helix in one of the high-resolution myoglobin structures. A convenient way to add 'HELIX', 'STRAND', or 'SHEET' restraints to the calculation by the 'model' script is to include them in the `special_restraints` routine (Section 1.9, Question 19). Note that at least the non-hydrogen mainchain atoms topology model is required although the same functionality could also be provided for the C_α -only topology with small changes to the source code.

- **'STRAND'**. This makes restraints enforcing an extended strand conformation for the residue segment specified by the two **RESIDUE_IDS** (Section 2.4.1). This is achieved by applying Φ, Ψ binormal restraints only. These binormal restraints force the mainchain conformation into class "B", except for the Pro residues which are restrained to class "P" [?].
- **'SHEET'**. This calculates H-bonding restraints for a pair of β -strands. **ATOM_IDS** specifies the two atom identifiers (Section 2.5.1) defining the first H-bond in the β -sheet ladder. **SHEET_H-BONDS** specifies the number of H-bonds to be added. The parallel and anti-parallel sheets are selected by a positive and negative integer in **SHEET_H-BONDS**, respectively. In a parallel sheet, hydrogen bonds start at the first or the second term of the following series (depending on **ATOM_IDS**): 1N:1O, 1O:3N, 3N:3O, 3O:5N, *etc.* For an anti-parallel sheet, the corresponding series is 1N:3O, 1O:3N, 3N:1O, 3O:1N, *etc.*; note that the residue indices are always decreasing for the second strand. The extended structure of the individual strands must be enforced separately by the **'STRAND'** restraints if so desired.
- **'USER_DISTANCE'**. This makes distance restraints between pairs of atoms from set 2 and 3 (inter-set only), using the value of **RESTRAINT_PARAMETERS**. Only distances satisfying the **RESIDUE_SPAN_RANGE** criterion are restrained. This command is useful for making non-specific "compactization" restraints.
- **Homology-derived restraints:**
 - **'DISTANCE'**. This makes distance restraints that are generated for all pairs of atoms i, j where atom i is from selected set 2 and atom j is from selected set 3 (as defined by the **PICK_ATOMS** command). The atoms also have to be within the residue spanning range specified by **RESIDUE_SPAN_RANGE = r1 r2**, such that the residue index difference $r1 \leq |ir2 - ir1| \leq r2$ when **RESIDUE_SPAN_SIGN = off** and $r1 \leq (ir2 - ir1) \leq r2$ when **RESIDUE_SPAN_SIGN = on**. Moreover, for a restraint to be created, at least one distance in the template structures must be less than **MAXIMAL_DISTANCE** (in Å). **RESTRAINT_STDEV = a b** specifies the linear transformation for the first six standard deviation models ($\sigma' = a + b * \sigma$) These six models are polynomials and depend on several structural features of the template and its similarity to the target. The polynomial coefficients are specified in library file **'\$PARAMS_LIB'**. When "polynomial model" 7 is selected, the standard deviation of restraints is set to constant a . Each basis pdf in the distance pdf corresponds to one template structure with an equivalent distance. The mean of this basis pdf is equal to the template distance and its standard deviation is calculated from an analytic model specified by **DISTANCE_RSR_MODEL**. Use model 5 for C_α - C_α distances and model 6 for N-O distances. The weights of basis pdf's depend on local sequence similarity between the target and the templates when **BASIS_PDF_WEIGHT = 'LOCAL'** and on global sequence identity when **BASIS_PDF_WEIGHT = 'GLOBAL'**. In addition, the atom pairs restrained by homology-derived restraints must by default not be in a chemical bond, chemical angle, dihedral angle, or on an excluded pairs list. This behavior can be changed by resetting **EXCL_LOCAL** (see **OPTIMIZE**).
 - **'PHI-PSI_CLASS', 'CHI1_DIHEDRAL', 'CHI2_DIHEDRAL', 'CHI3_DIHEDRAL', 'CHI4_DIHEDRAL', 'PHI-DIHEDRAL', 'PSI_DIHEDRAL', 'OMEGA_DIHEDRAL', 'PHI-PSI_BINORMAL'** are the mainchain and sidechain dihedral angle restraints. Only those dihedral angles are restrained that have all or at least **NON-BONDED_SEL_ATOMS** in the selected atom set 1. The means and standard deviations for the dihedral Gaussian restraints are obtained from the **\$RESDIH_LIB** and **\$MNCH?_LIB** libraries and their weights from the MDT tables, specified by **MDT_LIB_FILE** and **BIN_LIB_FILE**. The large MDT tables give the conditional weights for each possible dihedral angle class, as a function of all possible combinations of features on which a particular class depends. If **DIH_LIB_ONLY** is 0N or there is no equivalent residue in any of the templates, the weights for the dihedral angle classes depend only on the residue type and are obtained from the **'\$RESDIH_LIB'** and **'\$MNCH?_LIB'** libraries; the **DIH_LIB_ONLY** argument allows one to force the calculation of the "homology-derived" mainchain and sidechain dihedral angle restraints that ignore template information. **BASIS_PDF_WEIGHT** has the same effect as for the distance pdf's. **MDT_LIB_FILE** and **BIN_LIB_FILE** have to be specified for all homology-derived restraints that depend on the MDT files, including all mainchain and sidechain dihedral angle restraints. When **MODELLER's 'OMEGA'** restraints are calculated, the currently existing restraints on atoms **'O C +N +CA'** in all residues are automatically deleted. These deleted restraints correspond to the *improper* dihedral angles involving the ω atoms. They are deleted because they could be "frustrated" by the new **'OMEGA'** restraints. No action is taken with regard to any of the previously existing, possibly duplicated dihedral angle restraints. Thus, to avoid restraint duplication, including that of the **'OMEGA'** restraints, call the **CONDENSE_RESTRAINTS** command after all the restraints are calculated.

BASIS_RELATIVE_WEIGHT is the cutoff for removing weak basis pdf's from poly-Gaussian feature pdf's: a basis pdf whose weight is less than the BASIS_RELATIVE_WEIGHT fraction of the largest weight is deleted.

Example:

```
# Example for: MAKE_RESTRAINTS, SPLINE_RESTRAINTS, WRITE_RESTRAINTS

# This will compare energies of bond length restraints expressed
# by harmonic potential and by cubic spline.

SET OUTPUT_CONTROL = 1 1 1 1 1

READ_TOPOLOGY FILE = '$(LIB)/top_heav.lib'
READ_PARAMETERS FILE = '$(LIB)/par.lib'
READ_MODEL FILE = '1fas', MODEL_SEGMENT = '1:' '61:'
SEQUENCE_TO_ALI ATOM_FILES = '1fas', ALIGN_CODES = '1fas'
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ATOM_FILES = ATOM_FILES '1fas.ini', ;
                ALIGN_CODES = ALIGN_CODES '1fas-ini'
GENERATE_TOPOLOGY SEQUENCE = '1fas-ini'
TRANSFER_XYZ
BUILD_MODEL INITIALIZE_XYZ = off
WRITE_MODEL FILE '1fas.ini'

MAKE_RESTRAINTS RESTRAINT_TYPE = 'bond'
WRITE_RESTRAINTS FILE = '1fas-1.rsr'
ENERGY DYNAMIC_SPHERE = off

SPLINE_RESTRAINTS SPLINE_RANGE = 5.0, SPLINE_DX = 0.005, SPLINE_SELECT = 3 1 1
CONDENSE_RESTRAINTS
WRITE_RESTRAINTS FILE = '1fas-2.rsr'
ENERGY
```

2.5.4 DEFINE_SYMMETRY — define similar segments

Options:

SYMMETRY_WEIGHT = {real :1}	1.0	the weight of the symmetry objective function term
ADD_SYMMETRY = {logical :2}	off on	whether to add segment pair, add atoms to segment pair

Description: This command allows defining pairs of segments that will be restrained to be the same during optimization of the objective function. This is achieved by adding the sum of squares of the differences between the equivalent distances (similar to distance RMS deviation) to the objective function being optimized, separately for each pair of segments defined by **DEFINE_SYMMETRY**. The value of this term is reported in the log file by the **ENERGY** command, which also reports the individual contributions to the term when **OUTPUT** contains word 'SYMMETRY'. In each call of the **DEFINE_SYMMETRY** command, the list of such segments is either initiated, extended by a new pair of segments, or the last defined pair of segments is extended by adding new atoms.

SYMMETRY_WEIGHT specifies the atomic weights to be used in the calculation of the symmetry term (Eq. 5.71).

The two segments correspond to the selected sets 2 and 3 (obtained by the **PICK_RESTRAINTS** command). They must have the same number of atoms.

A pair of segments can be either added to the list (ADD_SYMMETRY[1] = on) or the list can be initialized (ADD_SYMMETRY[1] = off).

If ADD_SYMMETRY[2] = on, the currently selected atoms are added to the last segment pair in the segment pairs list, otherwise a new segment pair is started.

Example:

```
# Example for: DEFINE_SYMMETRY

# This will force two copies of 1fas to have similar mainchain
# conformation.

DEFINE_STRING VARIABLES = SEG1 SEG2

SET OUTPUT_CONTROL = 1 1 1 1 0

READ_TOPOLOGY FILE = '$(LIB)/top_heav.lib'
READ_PARAMETERS FILE = '$(LIB)/par.lib'

# Generate two copies of a segment:
READ_MODEL FILE = '2abx', MODEL_SEGMENT = '1:A' '74:B'
SEQUENCE_TO_ALI ALIGN_CODES = '2abx', ATOM_FILES = ALIGN_CODES
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ALIGN_CODES = ALIGN_CODES '2abx_ini', ;
    ATOM_FILES = ALIGN_CODES
GENERATE_TOPOLOGY SEQUENCE = '2abx_ini'
TRANSFER_XYZ
BUILD_MODEL INITIALIZE_XYZ = off
RENAME_SEGMENTS SEGMENT_IDS = 'A' 'B', RENUMBER_RESIDUES = 1 1
ENERGY DYNAMIC_SPHERE = off
RANDOMIZE_XYZ DEVIATION = 6.0
# Define the two segments (chains in this case) to be identical:
CALL ROUTINE = 'defsym', SEG1 = '1:A' '74:A', SEG2 = '1:B' '74:B'

# Make them identical by optimizing the initial randomized structure
# without any other restraints:
ENERGY
WRITE_MODEL FILE = 'define_symmetry-1.atm'
OPTIMIZE MAX_ITERATIONS = 300
WRITE_MODEL FILE = 'define_symmetry-2.atm'
ENERGY

# Now optimize with stereochemical restraints so that the
# result is not so distorted a structure (still distorted
# because optimization is not thorough):
SET DYNAMIC_SPHERE = on
MAKE_RESTRAINTS RESTRAINT_TYPE = 'stereo'
RANDOMIZE_XYZ DEVIATION = 3.0
SET MAX_ITERATIONS = 300, MD_RETURN = 'FINAL'
OPTIMIZE OPTIMIZATION_METHOD = 1 # Conjugate gradients
OPTIMIZE OPTIMIZATION_METHOD = 3 # Molecular dynamics
OPTIMIZE OPTIMIZATION_METHOD = 1 # Conjugate gradients
WRITE_MODEL FILE = 'define_symmetry-3.atm'
ENERGY

DELETE_ALIGNMENT
READ_MODEL MODEL_SEGMENT = '1:A' '74:A'
```

```

READ_MODEL2 MODEL2_SEGMENT = '1:B' '74:B'
PICK_ATOMS ATOM_TYPES = 'MNCH'
SUPERPOSE

STOP

SUBROUTINE ROUTINE = 'defsym'
  SET ATOM_TYPES = 'MNCH'
  SET SELECTION_STATUS = 'INITIALIZE'
  SET SELECTION_SEARCH = 'SEGMENT'

  SET SYMMETRY_WEIGHT = 1.0
  PICK_ATOMS PICK_ATOMS_SET = 2, SELECTION_SEGMENT = SEG1
  PICK_ATOMS PICK_ATOMS_SET = 3, SELECTION_SEGMENT = SEG2
  DEFINE_SYMMETRY ADD_SYMMETRY = on off

  RETURN
END_SUBROUTINE

```

2.5.5 PICK_RESTRAINTS — pick restraints for selected atoms

Options:

RESIDUE_RANGE = TYPEVALUES	DEFAULT	DESCRIPTION
RESTRAINTS_FILTER = {real : 33}	-999 -999 -999 -999	keep restraints?
	-999 -999 -999 -999	
	-999 -999 -999 -999	
	-999 -999 -999 -999	
	-999 -999 -999 -999	
	-999 -999 -999 -999	
	-999 -999 -999 -999	
	-999 -999 -999 -999	
	-999	
RESTRAINT_SEL_ATOMS = {integer : 1}	1	a restraint other than non-bonded pair has to have at least as many selected atoms
ADD_RESTRAINTS = {logical : 1}	off	whether to add new restraints to existing restraints

Description: This command selects some or all of the restraints currently in memory.

If `ADD_RESTRAINTS` is `on`, the already selected restraints remain selected; additional restraints also become selected if they satisfy currently specified conditions (see below). If `ADD_RESTRAINTS` is `off`, only those restraints that satisfy currently specified conditions become selected. This command runs over all restraints in memory, including the currently unselected restraints. Be careful about this: If you have some unselected restraints in memory, `PICK_RESTRAINTS` may select them; to prevent this, do `CONDENSE_RESTRAINTS` before calling `PICK_RESTRAINTS`.

A static restraint is selected if all or at least `RESTRAINT_SEL_ATOMS` of its atoms are selected (set 1), if it is strong enough based on its standard deviations or force constants (see the next paragraph), and if it does not span less (more) than the minimal (maximal) allowed number of residues specified by `RESIDUE_RANGE`. Note that here the `RESTRAINT_SEL_ATOMS` is used also for the static non-bonded restraints, while `MAKE_RESTRAINTS` and `OPTIMIZE` commands use `NONBONDED_SEL_ATOMS` for this purpose (`RESTRAINT_SEL_ATOMS` is used in `MAKE_RESTRAINTS` only for most restraint type other than non-bonded pairs).

To decide if a restraint is strong enough, the current standard deviations or force constants are compared with the corresponding `RESTRAINTS_FILTER`[`physical_restraint_type`]. A harmonic restraint, lower and upper bounds, and multi-modal Gaussian restraints are selected if the (smallest) standard deviation is less than the corresponding `RESTRAINTS_FILTER`[*i*]. The cosine energy term is selected if its force constant is larger than the corresponding `RESTRAINTS_FILTER`[*i*]. If `RESTRAINTS_FILTER`[*i*] = -999, a restraint of type *i* is always selected. Restraints of the other `physical_restraint_types` are always selected (Coulomb, Lennard-Jones, binormal, and spline). The `RESTRAINTS_FILTER` angles have to be specified in radians.

Example:

```
# Example for: PICK_RESTRAINTS, CONDENSE_RESTRAINTS

# This will pick only restraints that include at least one
# CA atom and write them to a file.

SET OUTPUT_CONTROL = 1 1 1 1 1

READ_TOPOLOGY FILE = '$(LIB)/top_heav.lib'
READ_PARAMETERS FILE = '$(LIB)/par.lib'
READ_MODEL FILE = '1fas'
SEQUENCE_TO_ALI ATOM_FILES = '1fas', ALIGN_CODES = '1fas'
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ATOM_FILES = ATOM_FILES '1fas.ini', ;
                ALIGN_CODES = ALIGN_CODES '1fas-ini'
GENERATE_TOPOLOGY SEQUENCE = '1fas-ini'
TRANSFER_XYZ
BUILD_MODEL INITIALIZE_XYZ = off

MAKE_RESTRAINTS RESTRAINT_TYPE = 'stereo'
ENERGY

PICK_ATOMS ATOM_TYPES = 'CA N C O'
PICK_RESTRAINTS ADD_RESTRAINTS = off, RESTRAINT_SEL_ATOMS = 1
# Delete the unselected restraints from memory:
CONDENSE_RESTRAINTS
ENERGY

WRITE_RESTRAINTS FILE = '1fas.rsr'
```

2.5.6 CONDENSE_RESTRAINTS — remove unselected restraints

Description: This command removes all the unselected restraints from memory. In addition, it also removes those cosine dihedral angle restraints (`RESTRAINT_TYPE = 'DIHEDRAL'`) that operate on the same atoms as any other restraints on a dihedral angle or a pair of dihedral angles. Such restraints include the `MODELLER` `'PHI_DIHEDRAL'`, `'PSI_DIHEDRAL'`, `'OMEGA_DIHEDRAL'`, `'CHI1_DIHEDRAL'`, `'CHI2_DIHEDRAL'`, `'CHI3_DIHEDRAL'`, `'CHI4_DIHEDRAL'`, `'PHI_PSI_CLASS'`, `'MRFP_DIHEDRAL'`, and `'PHI_PSI_BINORMAL'` dihedral angle restraints, as well as the 2nd, 3rd, *etc.* cosine dihedral angle restraints on the same atoms; the improper dihedral angle restraints are not considered here. For this command to work properly, the cosine dihedral angle restraints must be constructed before any other dihedral angle restraints. This functionality is needed because some of the CHARMM cosine terms are sometimes duplicated by other CHARMM cosine terms as well as by `MODELLER` homology-derived mainchain and sidechain dihedral and bi-dihedral angle restraints. In the standard `__model` script, the redundant CHARMM terms are always removed.

Example: See `READ_MODEL` command.

2.5.7 ADD_RESTRAINT — add restraint

Options:

```

ATOM_IDS = (string:0)          ''          atom ids: 'atom:residue_id[:chain_id]'
RESTRAINT_PARAMETERS = (real:0) 3 1 3 3 4 2 0 0.0 0.087  restraint parameters

```

Description: This command adds a specified restraint to the end of the restraints list and selects it. It can also add an excluded pair or a pseudo atom definition to the respective lists, depending on the dimension of RESTRAINT_PARAMETERS (Section 2.5.1). This command is useful for specifying *cis*-peptide bonds from a TOP script. The angles have to be in radians.

Example:

```

# Example for: ADD_RESTRAINT, DELETE_RESTRAINT

# This will enforce cis conformation for Pro-56.

# Make a model and stereochemical restraints:
SET OUTPUT_CONTROL = 1 1 1 1 0

DEFINE_STRING VARIABLES = ATOM_IDS1 ATOM_IDS2
READ_TOPOLOGY FILE = '$(LIB)/top_heav.lib'
READ_PARAMETERS FILE = '$(LIB)/par.lib'
READ_MODEL FILE = '1fas'
SEQUENCE_TO_ALI ATOM_FILES = '1fas', ALIGN_CODES = '1fas'
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ATOM_FILES = ATOM_FILES '1fas.ini', ;
                    ALIGN_CODES = ALIGN_CODES '1fas-ini'
GENERATE_TOPOLOGY SEQUENCE = '1fas-ini'
TRANSFER_XYZ
BUILD_MODEL INITIALIZE_XYZ = off
MAKE_RESTRAINTS RESTRAINT_TYPE = 'stereo'

# Change the Pro-56 restraint from trans to cis:
CALL ROUTINE = 'cispeptide', ATOM_IDS1 = 'O:56' 'C:56' 'N:57' 'CA:57', ;
                    ATOM_IDS2 = 'CA:56' 'C:56' 'N:57' 'CA:57'
WRITE_RESTRAINTS FILE = '1fas.rsr'
ENERGY

SUBROUTINE ROUTINE = 'cispeptide'
# Delete the old restraint on the same atoms:
DELETE_RESTRAINT ATOM_IDS = ATOM_IDS1
# Add the new restraint:
ADD_RESTRAINT RESTRAINT_PARAMETERS = 3 1 3 3 4 2 0 3.141593 0.087

DELETE_RESTRAINT ATOM_IDS = ATOM_IDS2
ADD_RESTRAINT RESTRAINT_PARAMETERS = 3 1 3 3 4 2 0 0.0 0.087

RETURN
END_SUBROUTINE

```

2.5.8 DELETE_RESTRAINT — unselect restraint

Options:

```
ATOM_IDS = (string : 0)          ''          atom ids: 'atom:residue_id[:chain_id]'
```

Requirements: MODEL

Description: This command scans the currently selected restraints to find all the restraints that operate on the specified atoms (Section 2.5.1) and then unselects them. The order of the atom names in ATOM_IDS does not matter: All restraints that contain all and only the specified atoms are unselected. This means that it is not possible to distinguish between the dihedral angle and improper dihedral angle restraints on the same four atoms.

The command only unselects the restraints found. To completely remove all the unselected restraints from memory, use **CONDENSE_RESTRAINTS**. The **DELETE_RESTRAINT** command is useful in specifying *cis*-peptide bonds from a TOP script.

Example: See **ADD_RESTRAINT** command.

2.5.9 REINDEX_RESTRAINTS — renumber MODEL2 restraints for MODEL

Requirements: restraints & MODEL & MODEL2

Description: This command renumbers atom indices in all restraints in memory. It is expected that the input restraints refer to MODEL2; the re-indexed restraints will correspond to MODEL. Both MODEL and MODEL2 have to be in memory. Only those restraints that have all atoms in MODEL will be selected. You can remove the others by **CONDENSE_RESTRAINTS**. This command is useful when the old restraints have to be used while changing from one topology model to another.

Example:

```
# Example for: REINDEX_RESTRAINTS

# This will reindex restraints obtained previously for a simpler topology so
# that they will now apply to a more complicated topology.

# Generate the model for the simpler topology (CA only in this case):
READ_TOPOLOGY FILE = '$(LIB)/top_ca.lib'
READ_PARAMETERS FILE = '$(LIB)/par_ca.lib'
SET TOPOLOGY_MODEL = 7
READ_MODEL FILE = '1fas'
SEQUENCE_TO_ALI ATOM_FILES = '1fas', ALIGN_CODES = '1fas'
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ATOM_FILES = ATOM_FILES '1fas.ca', ;
                ALIGN_CODES = ALIGN_CODES '1fas-ca'
GENERATE_TOPOLOGY SEQUENCE = '1fas-ca'
TRANSFER_XYZ
BUILD_MODEL INITIALIZE_XYZ = off
WRITE_MODEL FILE = '1fas.ca'

# Generate the restraints for the simpler topology:
MAKE_RESTRAINTS RESTRAINT_TYPE = 'stereo'
WRITE_RESTRAINTS FILE = '1fas-ca.rsr'
ENERGY

# Generate the model for the more complicated topology:
```

```

READ_TOPOLOGY FILE = '$(LIB)/top_heav.lib'
READ_PARAMETERS FILE = '$(LIB)/par.lib'
SET TOPOLOGY_MODEL = 3
READ_MODEL FILE = '1fas'
SET ADD_SEQUENCE = off
SEQUENCE_TO_ALI ATOM_FILES = '1fas', ALIGN_CODES = '1fas'
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ATOM_FILES = ATOM_FILES '1fas.ini', ;
        ALIGN_CODES = ALIGN_CODES '1fas-ini'
GENERATE_TOPOLOGY SEQUENCE = '1fas-ini'
TRANSFER_XYZ
WRITE_MODEL FILE = '1fas.ini'

READ_MODEL2 FILE = '1fas.ca'
REINDEX_RESTRAINTS
WRITE_RESTRAINTS FILE = '1fas.rsr'
ENERGY

```

2.5.10 SPLINE_RESTRAINTS — approximate restraints by splines

Options:

SPLINE_DX = \langle real : 1 \rangle	0.5	interval size for splining restraints
SPLINE_MIN_POINTS = \langle integer : 1 \rangle	5	have at least as many intervals in a spline
SPLINE_RANGE = \langle real : 1 \rangle	4.0	range of the splines
SPLINE_SELECT = \langle integer : 3 \rangle	4 1 9	specification of the restraints to be splined: form feature group

The **ENERGY** command keywords

Description: This command calculates and selects new restraints that are a spline approximation of the selected restraints of the specified type. It unselects the approximated restraints.

The type of the approximated restraints is specified by **SPLINE_SELECT** and is defined by the mathematical form (Gaussian, *etc*), feature type (distance, *etc*), and physical restraint group (sidechain χ_1 , *etc*) (the first, third, and fourth integer numbers in the restraint specification).

The restraint is approximated in a certain range only, determined differently for different mathematical forms. For example, the poly-Gaussian range is from $m - \text{SPLINE_RANGE} \times \sigma_m$ to $M + \text{SPLINE_RANGE} \times \sigma_M$, where m and M are the minimal and maximal means of the basis pdfs, and σ_m and σ_M are their corresponding standard deviations.

The spline points are distributed evenly over this range with an interval of **SPLINE_DX**. **SPLINE_DX** should be equal to the scale of the peaks of the restraint that you want to approximate reliably. The value of the restraint beyond the range is determined by linear extrapolation using the first derivatives at the bounds.

If the x-range and **SPLINE_DX** are such that the number of spline points would be less than **SPLINE_MIN_POINTS**, **SPLINE_DX** is decreased so that there are **SPLINE_MIN_POINTS** defining the “splined” restraint.

Example: See **MAKE_RESTRAINTS** command.

2.5.11 READ_RESTRAINTS — read spatial restraints

Options:

FILE = \langle string : 1 \rangle	'default'	input restraints file
DIRECTORY = \langle string : 1 \rangle	''	directory list (e.g., 'dir1:dir2:dir3:./:/')
FILE_FORMAT = TYPEVALUES	DEFAULT	DESCRIPTION

ADD_RESTRAINTS = \langle logical : 1 \rangle **off** whether to add new restraints to existing restraints

Description: This command reads restraints, excluded atom pairs, and pseudo atom definitions from a file. An excluded atom pair specifies two atoms that are not to be tested during generation of the dynamic non-bonded pair list. There is one restraint entry per line. The two possible formats of the file, **MODELLER** and **USER**, are described in Section 2.5. The routine determines automatically which format is used. The new restraints are added to those that are already in memory if **ADD_RESTRAINTS** = **on**, otherwise they initiate the restraints list. All the new restraints are automatically selected.

Example: See **MAKE_RESTRAINTS** command.

2.5.12 WRITE_RESTRAINTS — write spatial restraints

Options:

FILE = \langle string : 1 \rangle	'default'	partial or complete filename
OUTPUT_DIRECTORY = \langle string : 1 \rangle	''	output directory
FILE_FORMAT = TYPEVALUES	DEFAULT	DESCRIPTION

Description: This command writes the currently selected restraints to a file in the **MODELLER** format (see Section 2.5). If **'UNFORMATTED'** format is selected the file is approximately one third of the **'FORMATTED'** size. Both formats can be read with the **READ_RESTRAINTS** command. The output in the **USER** format is not yet implemented.

Example: See **MAKE_RESTRAINTS** command.

2.6 Optimization of the model

This section describes commands for creating, reading and writing optimization schedule, and for calculating and optimizing the objective function. For technical background, see Section 5.2.

2.6.1 MAKE_SCHEDULE — create optimization schedule

Options:

```
LIBRARY_SCHEDULE = <integer : 1>      1          selects schedule from the $SCHED_LIB library
SCHEDULE_SCALE = <real : 33>          1 1 1 1 1 1 1 1 1 1  factors for physical restraint types in scaling the
                                     1 1 1 1 1 1 1 1 1 1  schedule
                                     1 1 1 1 1 1 1 1 1 1
                                     1 1 1
```

Requirements: MODEL

Output: N_SCHEDULE

Description: This command constructs an optimization schedule for the variable target function method for the current MODEL.

The template for construction of the schedule is the LIBRARY_SCHEDULE-th entry in library file \$SCHED_LIB.

The usual schedule for the variable target function part of optimization in comparative modeling is as follows. The residue range (**PICK_RESTRAINTS** and Section 2.5.3) is increased with increasingly larger steps until the protein length is reached. The scaling of homology-derived and bonded stereochemical restraints increases from a small value to 1 in the initial few steps to allow for imperfect starting geometries, especially those that result from **RANDOMIZE_XYZ** and long insertions or deletions. The soft-sphere overlap restraints are slowly introduced only in the last four steps of the variable target function method to save CPU time and increase the radius of convergence. In comparative modeling by the 'model' script in the default mode, the variable target function method is usually followed by simulated annealing with molecular dynamics. In this last stage, all homology-derived and stereochemical restraints are generally used with the scaling factors of 1. There are a number of variables defined in the 'modlib/_defs.top' script that can be used to influence the thoroughness of both the variable target function and molecular dynamics parts of the optimization (Chapter 3).

The scaling factors for all physical restraint groups, in all schedule steps, are multiplied by the corresponding scalar in SCHEDULE_SCALE (1 by default). This is useful when template-derived fold restraints have to be weakened relative to some external restraints, so that the fold can actually reflect these external restraints, even when they are quite different from the template-derived restraints.

This command is an alternative to the **READ_SCHEDULE** command.

Use the **WRITE_SCHEDULE** command to find out what the calculated schedule is. The schedule file written by the 'model' routine has an extension .sch.

Example:

```
# Example for: MAKE_SCHEDULE, WRITE_SCHEDULE, READ_SCHEDULE

# This will create an VTFM optimization schedule for a model
# and write it to a file.

# MODEL has to be in memory for MAKE_SCHEDULE:
READ_MODEL FILE = '1fas'
MAKE_SCHEDULE LIBRARY_SCHEDULE = 1
# Write the schedule to a file:
WRITE_SCHEDULE FILE = '1fas.sch'
# Read it in just for fun:
READ_SCHEDULE FILE = '1fas.sch'
```

2.6.2 READ_SCHEDULE — read optimization schedule

Options:

FILE = \langle string : 1 \rangle	'default'	partial or complete filename
DIRECTORY = \langle string : 1 \rangle	''	directory list (e.g., 'dir1:dir2:dir3:./:')
SCHEDULE_SCALE = \langle real : 33 \rangle	1 1 1 1 1 1 1 1 1 1	factors for physical restraint types in scaling the
	1 1 1 1 1 1 1 1 1 1	schedule
	1 1 1 1 1 1 1 1 1 1	
	1 1 1	

Output: N_SCHEDULE

Description: This command reads a text file that contains an optimization schedule for the variable target function method.

Each line in the file contains in free format the parameters for a single step of the variable target function method. These parameters are: step index (not used by the program), optimization method, maximal difference in residue indices of atoms restrained by the selected restraints (**PICK_RESTRAINTS** and Section 2.5.3), and the scaling factors for all types of restraints. The smaller the scaling factor, the weaker the corresponding restraint.

See **MAKE_SCHEDULE** for explanation of SCHEDULE_SCALE.

This command also sets the TOP variable N_SCHEDULE to the total number of the variable target function steps that were read in.

Example: See **MAKE_SCHEDULE** command.

2.6.3 WRITE_SCHEDULE — write optimization schedule

Options:

FILE = \langle string : 1 \rangle	'default'	partial or complete filename
OUTPUT_DIRECTORY = \langle string : 1 \rangle	''	output directory

Description: This command writes out the schedule for the variable target function method. This schedule file can then be read by the **READ_SCHEDULE** command.

Example: See **MAKE_SCHEDULE** command.

2.6.4 ENERGY — evaluate MODEL given restraints

Options:

VIOL_REPORT_CUT = <code><real : 33></code>	4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 999 999 999 999 4.5 4.5 4.5 4.5 4.5 4.5 999 6.5 4.5 4.5 4.5 4.5 4.5 999 999 999	cutoffs for reporting relative violations
VIOL_REPORT_CUT2 = <code><real : 33></code>	2.0 2.0	
OUTPUT = <code><string : 1></code>	'LONG'	'SHORT' 'LONG' 'VERY_LONG' 'GRADIENT' 'SYMMETRY' 'ENERGY_PROFILE' 'VIOLATIONS_PROFILE'
NORMALIZE_PROFILE = <code><logical : 1></code>	off	whether to normalize energy/violations profiles or not, by the number of terms per residue
SMOOTHING_WINDOW = <code><integer : 1></code>	3	profiles are smoothed over 2*SW + 1 residues
ASGL_OUTPUT = <code><logical : 1></code>	off	whether to write output for ASGL
SCHEDULE_STEP = <code><integer : 1></code>	1	schedule step for optimization
TOPOLOGY_MODEL = <code><integer : 1></code>	3	selects topology library: 1-9
RADII_FACTOR = <code><real : 1></code>	0.82	factor for van der Waals radii
SPHERE_STDV = <code><real : 1></code>	0.05	standard deviation of soft-sphere repulsion
DYNAMIC_SPHERE = <code><logical : 1></code>	on	whether to use dynamic soft-sphere repulsion terms
DYNAMIC_LENNARD = <code><logical : 1></code>	off	whether to use dynamic Lennard-Jones energy terms
DYNAMIC_COULOMB = <code><logical : 1></code>	off	whether to use dynamic Coulomb energy terms
DYNAMIC_MODELLER = <code><logical : 1></code>	off	whether to use dynamic MODELLER non-bonded restraints
DYNAMIC_ACCESS = <code><logical : 1></code>	off	whether to use dynamic accessibility energy terms
EXCL_LOCAL = <code><logical : 4></code>	on on on on	whether to exclude bonds, angles, dihedrals, explicit excl pairs from the homology-derived distance rsrs
LENNARD_JONES_SWITCH = <code><real : 2></code>	6.5 7.5	the range for Lennard-Jones interaction smoothing to 0
COULOMB_SWITCH = <code><real : 2></code>	6.5 7.5	the range for Coulomb interaction smoothing to 0
RELATIVE_DIELECTRIC = <code><real : 1></code>	1.0	relative dielectric constant
CONTACT_SHELL = <code><real : 1></code>	4.0	distance cutoff for calculation of the non-bonded pairs list
UPDATE_DYNAMIC = <code><real : 1></code>	0.39	when to update non-bonded pairs list
NLOGN_USE = <code><integer : 1></code>	15	number of residues at which to begin using the N Log N non-bonded pairs routine
COVALENT_CYS = <code><logical : 1></code>	off	whether to consider SG-SG covalent bond similar to polypeptide chain when proximity of residues along the sequence is considered. If PATCH_SS_MODEL is done, then make it ON.

```
RESIDUE_SPAN_RANGE = (integer : 2)  0 99999          range of residues spanning the allowed
                                                distances; for MAKE_RESTRAINTS,
                                                PICK_RESTRAINTS, non-bonded dynamic
                                                pairs
```

Output: MOLPDF**Requirements:** MODEL & restraints

Description: The main purpose of this command is to compare spatial features of the current MODEL with the selected restraints in order to determine the violations of the molecular pdf. It lists variable amounts of information about the values of the basis, feature, and molecular pdf's for the current MODEL. All arguments that affect the value of the molecular pdf are also relevant for the **ENERGY** command.

Most of the output goes to the log file. The output of the **ENERGY** command has to be examined carefully, at least at the end of the optimization, when the final model is produced. Additional output files, for the ASGL plotting program are created if ASGL_OUTPUT = on (undocumented).

OUTPUT selects various kinds of output information:

- 'LONG' writes restraint violations one per line to the log file.
- 'VERY_LONG' writes the most detailed examination of the selected basis and feature pdf's to the log file, using several lines of output for each restraint.
- 'GRADIENT' writes the 'force' gradients for the currently selected restraints to the isotropic temperature factors for each atom of the current MODEL.
- 'SYMMETRY' writes a comparison of equivalent distances involved in the definition of the symmetry enforcing term to the log file.

VIOL_REPORT_CUT is a vector with one real number for each physical restraint type. A restraint is reported when its 'heavy relative violation' is larger than the corresponding cutoff. The heavy relative violation is calculated by finding the global minimum of a feature according to the restraint, taking the difference between the actual feature in the model and this global minimum, and then normalizing the difference by the standard deviation of the global minimum. The 'minimal violation' of a restraint is defined as the difference from the local minimum closest to the value of the feature in the model (with the exception of the spline restraints; see next paragraph).

VIOL_REPORT_CUT2 is similar to VIOL_REPORT_CUT, except that it contains cutoffs for restraint 'energies', not heavy relative violations.

The meaning of various other reported properties of the violated restraints is briefly described in the log file. Note that for multi-modal restraints that are described by cubic splines (by default, all multimodal homology-derived restraints), only one optimal value is defined, not the local and global minimum as for the multi-modal Gaussian restraints. As a result, the minimal violations and heaviest violations are the same. For interpreting the seriousness of violations, use the following rule of thumb: There should be at most a few small violations (*e.g.*, 4 standard deviations) for all monomodal restraints. In comparative modeling, the monomodal restraints include the stereochemical restraints and distance restraints when only one homologous structure is used. For the multimodal restraints, there are usually many violations reported because the heaviest violations are used in deciding whether or not to report a violation. In comparative modeling, the multimodal restraints include the χ_i restraints, (Φ , Ψ) binormal restraints and distance restraints when more than one template is used. See also Section 1.9, Question 22.

For profiles:

This command calculates residue energies or heavy relative violations, depending on OUTPUT, for all physical restraint types (there are NPHYCNS of them). Relative heavy violations (Table 2.2) are used because only *relative* violations of different features are comparable. In both cases, the residue sum is the sum over all restraints that have at least one atom in a given residue. The contribution of each restraint is counted exactly once for each residue, without any weighting. Restraints spanning more than one residue contribute equally to all of them. Thus, the sum of residue energies is generally larger than molecular pdf. The command also

calculates the sum of the NPHYCNS contributions for each residue and writes all NPHYCNS+1 columns to a file suitable for plotting by ASGL.

If NORMALIZE_PROFILE is on the profile for each residue is normalized by the number of terms applying to each residue.

All the curves are smoothed by the running window averaging method if SMOOTHING_WINDOW is larger than 0: The window is centered on residue i and extends for $(SMOOTHING_WINDOW/2) - 1$ residues on each side. Thus, SMOOTHING_WINDOW has to be an even number (or it is made such by the program automatically). The only exceptions are the two termini, where a smaller number of residues are available for smoothing. The relative weight of residue j when calculating the smoothed value at residue i is $(SMOOTHING_WINDOW/2 - |j - i|)$.

The energy or the violations profile is written to the fourth column of the MODEL atomic records (atomic isotropic temperature factors for X-ray structures). Note that all the atoms in one residue get the same number. This output is useful for exploring the violations on a graphics terminal.

See description of OPTIMIZE for the other variables.

Example:

```
# Example for: ENERGY

# This will calculate the stereochemical energy (bonds,
# angles, dihedrals, impropers) for a given model.

READ_TOPOLOGY FILE = '$(LIB)/top_heav.lib'
READ_PARAMETERS FILE = '$(LIB)/par.lib'
READ_MODEL FILE = '1fas'
SEQUENCE_TO_ALI ATOM_FILES = '1fas', ALIGN_CODES = '1fas'
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ATOM_FILES = ATOM_FILES '1fas.ini', ;
                ALIGN_CODES = ALIGN_CODES '1fas-ini'
GENERATE_TOPOLOGY SEQUENCE = '1fas-ini'
# Must patch disulfides here to calculate the non-bonded
# energy properly. Also, when you use hydrogens, disulfides
# must always be patched so that sulfhydryl hydrogens are
# removed from the model.
PATCH RESIDUE_TYPE = DISU, RESIDUE_IDS = '17' '39'
PATCH RESIDUE_TYPE = DISU, RESIDUE_IDS = '3' '22'
PATCH RESIDUE_TYPE = DISU, RESIDUE_IDS = '53' '59'
PATCH RESIDUE_TYPE = DISU, RESIDUE_IDS = '41' '52'
TRANSFER_XYZ
BUILD_MODEL INITIALIZE_XYZ = off

MAKE_RESTRAINTS RESTRAINT_TYPE = 'stereo'
ENERGY DYNAMIC_SPHERE = on
```

2.6.5 OPTIMIZE — optimize MODEL given restraints

Options:

OPTIMIZATION_METHOD = \langle integer : 1 \rangle	-999	type of optimization method: 1 3
SCHEDULE_STEP = \langle integer : 1 \rangle	1	schedule step for optimization
TOPOLOGY_MODEL = \langle integer : 1 \rangle	3	selects topology library: 1-9
RADII_FACTOR = \langle real : 1 \rangle	0.82	factor for van der Waals radii
SPHERE_STDV = \langle real : 1 \rangle	0.05	standard deviation of soft-sphere repulsion

DYNAMIC_SPHERE = \langle logical : 1 \rangle	on	whether to use dynamic soft-sphere repulsion terms
DYNAMIC_LENNARD = \langle logical : 1 \rangle	off	whether to use dynamic Lennard-Jones energy terms
DYNAMIC_COULOMB = \langle logical : 1 \rangle	off	whether to use dynamic Coulomb energy terms
DYNAMIC_MODELLEER = \langle logical : 1 \rangle	off	whether to use dynamic MODELLER non-bonded restraints
DYNAMIC_ACCESS = \langle logical : 1 \rangle	off	whether to use dynamic accessibility energy terms
EXCL_LOCAL = \langle logical : 4 \rangle	on on on on	whether to exclude bonds, angles, dihedrals, explicit excl pairs from the homology-derived distance rsrs
LENNARD_JONES_SWITCH = \langle real : 2 \rangle	6.5 7.5	the range for Lennard-Jones interaction smoothing to 0
COULOMB_SWITCH = \langle real : 2 \rangle	6.5 7.5	the range for Coulomb interaction smoothing to 0
RELATIVE_DIELECTRIC = \langle real : 1 \rangle	1.0	relative dielectric constant
NONBONDED_SEL_ATOMS = \langle integer : 1 \rangle	1	a non-bonded pair has to have at least as many selected atoms
RESIDUE_SPAN_RANGE = \langle integer : 2 \rangle	0 99999	range of residues spanning the allowed distances; for MAKE_RESTRAINTS, PICK_RESTRAINTS, non-bonded dynamic pairs
COVALENT_CYS = \langle logical : 1 \rangle	off	whether to consider SG-SG covalent bond similar to polypeptide chain when proximity of residues along the sequence is considered. If PATCH_SS_MODEL is done, then make it ON.
CONTACT_SHELL = \langle real : 1 \rangle	4.0	distance cutoff for calculation of the non-bonded pairs list
UPDATE_DYNAMIC = \langle real : 1 \rangle	0.39	when to update non-bonded pairs list
NLOGN_USE = \langle integer : 1 \rangle	15	number of residues at which to begin using the N Log N non-bonded pairs routine
TRACE_OUTPUT = \langle integer : 1 \rangle	0	modulus for writing information about optimization iterations: 0 for nothing
MAX_ITERATIONS = \langle integer : 1 \rangle	200	maximal iterations in optimization
OUTPUT = \langle string : 1 \rangle	'LONG'	'NO_REPORT'
• For conjugate gradients: MIN_ATOM_SHIFTS = TYPEVALUES	DEFAULT	'NO_REPORT' 'REPORT'
• For molecular dynamics: MD_TIME_STEP = \langle real : 1 \rangle	4.0	time step for MD in fs
INIT_VELOCITIES = \langle logical : 1 \rangle	on	whether to initialize velocities before MD
TEMPERATURE = \langle real : 1 \rangle	293.0	temperature for MD simulation in K
EQUILIBRATE = \langle integer : 1 \rangle	999999	equilibrate during MD every that many steps
MD_RETURN = \langle string : 1 \rangle	'FINAL'	return MODEL with 'MINIMAL' energy or 'FINAL' MODEL
CAP_ATOM_SHIFT = \langle real : 1 \rangle	0.2	limit for atomic shifts in optimization
RAND_SEED = \langle integer : 1 \rangle	-8123	random seed from -50000 to -2
STOP_ON_ERROR = \langle integer : 1 \rangle	1	whether to stop on error

Output: MOLPDF, MODELLER_STATUS

Requirements: MODEL & restraints

Description: This command performs a number of optimizing iterations using a selected optimization method (5.2). One call to **OPTIMIZE** corresponds to a single step of the variable target function method. The

whole variable target function method is implemented by a TOP script. The molecular pdf is optimized with respect to the selected coordinates of the current MODEL; the optimized coordinates are returned as the current MODEL.

Some output may be generated during optimization; for example, a value of the molecular pdf, average and maximal atomic shifts are written to the current tracing file every TRACE_OUTPUT iterations of the optimizer if TRACE_OUTPUT is larger than 0 (see the SWITCH_TRACE command).

In addition, a summary of the optimization results is written to the log file after optimization, unless OUTPUT contains string 'NO_REPORT'.

OPTIMIZATION_METHOD = 1 selects a conjugate gradients optimization method. OPTIMIZATION_METHOD = 3 selects a molecular dynamics optimization at a fixed temperature. The conjugate gradients optimizer is a modified version of the Beale restart conjugate gradients method [?, ?]. The molecular dynamics routine is the most basic version of the iterative solver of the Newton's equations of motion. The integrator uses the Verlet algorithm [?]. All atomic masses are set to that of carbon 12. A brief description of the algorithms is given in Section 5.2.

SCHEDULE_STEP is the variable target function step. It selects some of the optimization parameters; it refers to the line in the schedule file which specifies (1) the optimization method (1=Conjugate Gradients, 3=Molecular Dynamics); (2) maximal number of residues that the restraints are allowed to span (Section 2.5.3); (3) the individual scaling factors for all the physical restraint types. OPTIMIZATION_METHOD overrides the schedule specification if it is within a defined range.

CONTACT_SHELL defines the maximal distance between atoms that flags a non-bonded atom pair. Such pairs are stored in the list of non-bonded atom pairs. Only those non-bonded pairs that are sufficiently close to each other will result in an actual non-bonded restraint. If undefined (-999), the default value is the maximum of the three possibilities: twice the radius of the largest atom multiplied by RADII_FACTOR (in the case of the all non-hydrogen atoms model, this is 3.2 Å); LENNARD_JONES_SWITCH[2]; or COULOMB_SWITCH[2]. Only those values of the three possibilities are compared that have the corresponding DYNAMIC_SPHERE, DYNAMIC_LENNARD, or DYNAMIC_COULOMB set to on. The best value for CONTACT_SHELL must be found in combination with UPDATE_DYNAMIC (see also below). Good values are 4Å for CONTACT_SHELL and 0.39Å for UPDATE_DYNAMIC when no Lennard-Jones and Coulomb terms are used; if CONTACT_SHELL is larger, there would be many pairs in the non-bonded pairs list which would slow down the evaluation of the molecular pdf. If it is too small, however, the increased frequency of the pair list recalculation may slow down the optimization. It is useful in some simulations to be able to set CONTACT_SHELL to something large (e.g., 8Å) and UPDATE_DYNAMIC to 999999.9, so that the pairs list is prepared only at the beginning. However, you have to make sure that the potential energy is not invisibly pumped into the system by making contacts that are not on the list of non-bonded pairs (see below).

UPDATE_DYNAMIC sets the cumulative maximal atomic shift that triggers recalculation of the list of atom-atom non-bonded pairs. It should be set in combination with CONTACT_SHELL. For soft-sphere overlap, to be absolutely sure that no unaccounted contacts occur, UPDATE_DYNAMIC has to be equal to (CONTACT_SHELL - maximal_overlap_distance) / 2. Maximal_overlap_distance is equal to the diameter of the largest atom in the model; it is 3.2 Å in the case of the all non-hydrogen atoms model. This distance is the CONTACT_SHELL value if a default is requested. Factor 2 comes from the fact that the moves of both atoms can reduce the distance between them. DYNAMIC_SPHERE has to be set to on for the automatic generation of the soft-sphere overlap restraints. Another necessary condition is that the scaled standard deviation of the soft-sphere overlap restraints is greater than zero. It is simpler not to pre-calculate any soft-sphere overlap restraints and to use the dynamically generated restraints alone, although this may be slower.

Similarly, DYNAMIC_LENNARD, DYNAMIC_COULOMB, DYNAMIC_MODELLER and DYNAMIC_ACCESS determine whether the dynamic Lennard-Jones terms, electrostatic interactions, MODELLER non-bonded spline restraints and MODELLER atomic density restraints are calculated during optimization. Currently, the first derivatives of the atom density restraints are set to 0. SHELL here xx.

EXCL_LOCAL[4] specifies whether or not the atoms in a chemical bond, chemical angle, dihedral/improper angle, and in the excluded pairs list are considered in the construction of the non-bonded atom pairs list. This is especially useful when simplified protein representations are used; e.g., when non-bonded restraints need to be used on $C_{\alpha i} - C_{\alpha i+2}$ terms.

The initial atom radii (before scaling by `RADII_FACTOR`) depend on `TOPOLOGY_MODEL` which selects a column of radii for the specified topology model from the `$RADII_LIB` library file.

`RADII_FACTOR` is the scaling factor for the atom radii as read from the library file. The scaled radii are used only for the calculation of violations of the soft-sphere overlap restraints.

`LENNARD_JONES_SWITCH` is a real vector of two elements. It specifies r_{min} and r_{max} for the Lennard-Jones interaction (Eq. 5.62). The potential is smoothed down to zero between these two distances.

`COULOMB_SWITCH` is a real vector of two elements. It specifies r_{min} and r_{max} for the electrostatic interaction (Eq. 5.59). The potential is smoothed down to zero between these two distances.

`RESIDUE_SPAN_RANGE` determines what atom pairs can possibly occur in the dynamic non-bonded atom pairs list (see `MAKE_RESTRAINTS`). `RESIDUE_SPAN_SIGN` is ignored in `OPTIMIZE`. The effect of `RESIDUE_SPAN_RANGE` is modulated by `COVALENT_CYS`. If `COVALENT_CYS` is on, the disulfide bridges are taken into account when calculating the residue index difference between two atoms (*i.e.*, disulfides make some atom pairs closer in sequence). `COVALENT_CYS = on` is slow and only has an effect when certain statistical non-bonded potentials are used (*i.e.*, `DYNAMIC_MODELLE` is on and the non-bonded library has been derived considering the disulfide effect). Thus, it should generally be set to `off`. The dynamic restraints include soft-sphere overlap, Lennard-Jones, electrostatic restraints, and general spline restraints. The first three types of restraints can also be generated as static restraints by `MAKE_RESTRAINTS`.

The automatically generated dynamic restraints are always deleted after a command that calculates them is finished (`OPTIMIZE`, `ENERGY`, `PICK_HOT_ATOMS`); you have to use `MAKE_RESTRAINTS` to calculate equivalent static restraints if you want to write the ‘dynamic’ restraints to a file.

`MIN_ATOM_SHIFT` is a convergence criterion for the conjugate gradients optimization. When the maximal atomic shift is less than the specified value, the optimization is finished regardless of the number of optimization cycles or function value and its change.

`MAX_ITERATIONS` is used to prevent a waste of CPU time in the conjugate gradients optimization. When that many cycles are done, the optimization is finished regardless of the maximal atomic shift.

Before calculating dynamic non-bonded restraints, `MODELLER` determines which of the several routines is most appropriate and efficient for calculating the non-bonded atom pairs list. The user can influence this selection by specifying two variables: `NONBONDED_SEL_ATOMS`, which has an effect when only a subset of all atoms is selected by the `PICK_ATOMS` or `PICK_HOT_ATOMS` commands (set 1), and `NLOGN_USE`, which has an effect when all atoms are selected. If `NONBONDED_SEL_ATOMS` is 2 (default), the non-bonded pairs will contain only selected atoms (set 1). This means that the optimized atoms will not “feel” the rest of the protein through the non-bonded terms at all. If `NONBONDED_SEL_ATOMS` is 1, only one of the atoms in the non-bonded pair has to be a selected atom. This means that the selected region feels the rest of the system through the non-bonded terms, at the expense of longer CPU times. When all atoms are selected, `NONBONDED_SEL_ATOMS` of course has no effect. However, in that case, `NLOGN_USE` is used to select either a straightforward $\mathcal{O}(n^2)$ search or a cell-based algorithm which has $n \log n$ dependency of CPU time *versus* size n . The latter algorithm is used when the maximal difference in residue indices of the atoms in the current dynamic restraints is larger than `NLOGN_USE` or when the box size for this algorithm would have to be larger than 8Å.

The molecular dynamics optimizer pretends that the natural logarithm of the molecular pdf is energy in kcal/mole. `MD_TIME_STEP` is the time step in femtoseconds. `TEMPERATURE` is the temperature of the system in degrees Kelvin. `MAX_ITERATIONS` determines the number of MD steps. If `MD_RETURN` is ‘FINAL’ the last structure is returned as the `MODEL`. If `MD_RETURN` is ‘MINIMAL’ then the structure with the lowest value of the objective function on the whole trajectory is returned as the `MODEL`. Rescaling of velocities is done every `EQUILIBRATION` steps to match the specified temperature. Atomic shifts along one axis are limited by `CAP_ATOM_SHIFT`. This value should be smaller than `UPDATE_DYNAMIC`. If `INIT_VELOCITIES = on`, the velocity arrays are initialized, otherwise they are not. In that case, the final velocities from the previous run are used as the initial velocities for the current run.

`RAND_SEED` is the seed for the random number generator. It has to be between -2 and -50000 . Its value is changed after the return from the optimization routine.

`MOLPDF` contains the value of the objective function at the end of optimization.

`MODELLER_STATUS` is set to 1 if optimization is aborted because dynamic restraints could not be calculated as a result of a system being too large. If `MODELLER_STATUS` is equal or greater than `STOP_ON_ERROR`

the execution is stopped. Otherwise the execution returns back to the TOP routine, exiting all optimization routines immediately. The execution then continues as if nothing happened. It is up to the calling TOP routine to ensure that sensible action is taken; *e.g.*, skipping the rest of modeling for the model that resulted in an impossible function evaluation. This option is useful when calculating several independent models and you do not want one bad model to abort the whole calculation. A probable reason for an interrupted optimization is that it was far from convergence by the time the calculation of dynamic restraints was first requested. Two possible solutions are: (1) optimize more thoroughly (*i.e.* slowly) and (2) use a different contact pairs routine (SET NLOGN_USE = 9999). MODELLER.STATUS can be used in the TOP routine to exit from an optimization of a hopeless model and to continue with another model from a different initial conformation.

Example:

```
# Example for: OPTIMIZE, SWITCH_TRACE

# This will optimize stereochemistry of a given model, including
# non-bonded contacts.

READ_TOPOLOGY FILE = '$(LIB)/top_heav.lib'
READ_PARAMETERS FILE = '$(LIB)/par.lib'
READ_MODEL FILE = '1fas'
SEQUENCE_TO_ALI ATOM_FILES = '1fas', ALIGN_CODES = '1fas'
SEQUENCE_TO_ALI ADD_SEQUENCE = on, ATOM_FILES = ATOM_FILES '1fas.ini', ;
                ALIGN_CODES = ALIGN_CODES '1fas-ini'
GENERATE_TOPOLOGY SEQUENCE = '1fas-ini'
TRANSFER_XYZ
BUILD_MODEL INITIALIZE_XYZ = off
WRITE_MODEL FILE = '1fas.ini'

# Generate the restraints:
MAKE_RESTRAINTS RESTRAINT_TYPE = 'stereo'
WRITE_RESTRAINTS FILE = '1fas.rsr'
ENERGY DYNAMIC_SPHERE = on
SWITCH_TRACE TRACE_OUTPUT = 1, FILE = '1fas.trc'
OPTIMIZE OPTIMIZATION_METHOD = 1, MAX_ITERATIONS = 20
OPTIMIZE OPTIMIZATION_METHOD = 3, TEMPERATURE = 300, MAX_ITERATIONS = 50
OPTIMIZE OPTIMIZATION_METHOD = 1, MAX_ITERATIONS = 20
ENERGY

WRITE_MODEL FILE = '1fas.B'
```

2.6.6 SWITCH_TRACE — open new optimization trace file

Options:

FILE = <string : 1>	'default'	partial or complete filename
DIRECTORY = <string : 1>	''	directory list (<i>e.g.</i> , 'dir1:dir2:dir3:./:/')
TRACE_OUTPUT = <integer : 1>	0	modulus for writing information about optimization iterations: 0 for nothing

Description: This command specifies the file for the subsequent optimization tracing output. It is useful for separating tracing output for different models constructed in a single run of MODELLER. The tracing output is only produced if TRACE_OUTPUT is larger than 0. The tracing file includes the iteration number, number

Column	Description
1	iteration number within one step of the variable target function method
2	number of function evaluations within one step of VTFM
3	objective function value
4	average atomic shift
5	maximal atomic shift
6	proportional to the gradient
7	kinetic energy
8	temperature for molecular dynamics optimization
9	total energy (kinetic and potential; potential = objective function value)

Table 2.5: *Columns in an optimization trace file.*

of function evaluations, function value, average and maximal atomic shifts, the size of the gradient vector, kinetic energy (for molecular dynamics ‘optimization’ only), temperature (MD only) and total energy. This is written out in every TRACE_OUTPUT-th cycle of whatever optimization method is used, starting with the state just before the optimization (iteration 0).

When using the model script for comparative modeling, there is one .D file for each .B file with a model. The .D files contain information about the progress of optimization, from the beginning to the end. The most important column is column 3, which contains the value of the objective function, which is being optimized, as a function of the iteration step (every 10 steps, by default). Thus, the best model, according to MODELLER, is the one that has the lowest number in the third column of the last line of its .D file. This value is also written out in the REMARK record of the PDB file containing the model and in the log file.

Example: See OPTIMIZE command.

2.6.7 DEBUG_FUNCTION — test code self-consistency

Options:

```

DEBUG_FUNCTION_CUTOFF = <real : 3> 0.01 0.001 0.1      cutoffs for reporting differences between numeri-
                                                         cal and analytical derivatives: absolute, relative
                                                         errors, factor_for_indiv_rstrs
DETAILED_DEBUGGING = <logical : 1>  off              whether to evaluate energy and derivatives wrt
                                                         each restraint

```

all the ENERGY options

Description: This command checks the self-consistency of the code for the objective function and its derivatives by calculating and comparing numeric and analytical derivatives. All the parameters influencing the evaluation of the molecular pdf are also relevant (see ENERGY). The derivative is reported if both the absolute difference and the fractional difference between the two kinds of evaluations are larger than DEBUG_FUNCTION_CUTOFF[1] and DEBUG_FUNCTION_CUTOFF[2], respectively.

When DETAILED_DEBUGGING is on, the analytic and numeric derivatives of each restraint with respect to atomic positions are also compared for the atoms ‘violated’ by the whole molecular pdf. The absolute cutoff for writing out the discrepancies is scaled by DEBUG_FUNCTION_CUTOFF[3]; the relative cutoff remains the same as before.

When MODELLER is compiled in double precision, this test reports a smaller number of discrepancies.

Example:

```
# Example for: DEBUG_FUNCTION
```

```
# This will use default MODELLER scripts to construct homology
# restraints for 1fas. It will then use DEBUG_FUNCTION to test
# the source code for the function and derivatives calculation
# by comparing analytical and numerical first derivatives.
# Some discrepancies will be reported but ignore them here.

INCLUDE

SET OUTPUT_CONTROL = 1 1 1 1 1

SET ALNFILE = 'debug_function.ali'
SET SEQUENCE = '1fas'
SET KNOWNNS = '2ctx' '1nbt'
SET SPLINE_ON_SITE = off
CALL ROUTINE = 'model', EXIT_STAGE = 1

# To assign 0 weights to restraints whose numerical derivatives
# code does not work (i.e., splines for angles and dihedrals):
READ_SCHEDULE FILE = 'debug_function.sched'
ENERGY
DEBUG_FUNCTION DEBUG_FUNCTION_CUTOFF = 15.00 0.10 0.1, DETAILED_DEBUGGING = on
```

Chapter 3

MODELLER scripts

This section describes some of the MODELLER scripts found in the `$MODINSTALL6/bin/__.top` files. All these files and brief descriptions are listed in Table 3.1.

Filename	Description
<code>__model.top</code>	the main script for comparative modeling with user alignment
<code>__full_homol.top</code>	the main script for comparative modeling with automatic alignment
<code>__loop.top</code>	loop modeling (in development!)
<code>__defs.top</code>	variable definitions for modeling by <code>model</code>
<code>__align_strs.seq.top</code>	aligning many structures with a sequence
<code>__getnames.top</code>	generating default filenames from protein codes
<code>__homcsr.top</code>	generating homology-derived restraints
<code>__spline.top</code>	generating splined restraints
<code>__cisperptide.top</code>	defining <i>cis</i> -peptides
<code>__default_patches.top</code>	making topology patches during modeling by <code>model</code>
<code>__special.top</code>	generating and reading special restraints/patches for modeling by <code>model</code>
<code>__generate_model.top</code>	generating initial models for modeling by <code>model</code>
<code>__single_model.top</code>	used by <code>model</code> to generate a single model
<code>__multiple_models.top</code>	used by <code>model</code> to generate an ensemble of models
<code>__refine.top</code>	molecular dynamics refinement for modeling by <code>model</code>
<code>__loop.top</code>	modeling of loops
<code>__sidehchain.top</code>	modeling of a point mutation
<code>__principal.top</code>	principal components clustering
<code>__cluster.top</code>	optimization by 'clustering' and refinement
<code>__spline.top</code>	spline most restraint types in memory
<code>__asgl_mod.top</code>	plotting for clustering analysis (requires ASGL)
<code>__complete.top</code>	generating missing atoms in a PDB file
<code>__fit.top</code>	superposing two structures, given an alignment
<code>__mod.top</code>	the main include file including all other <code>__.top</code> files

Table 3.1: *List of MODELLER scripts.*

3.1 Flowchart of comparative modeling by MODELLER

This section describes a flowchart of comparative modeling by MODELLER, as implemented in the 'model' TOP script. This script is also called by QUANTA and INSIGHTII. It can be used for a variety of modeling tasks, not only for comparative modeling.

Input: script file, alignment file, PDB file(s) for template(s).

Output:

```

job.log          log file
job.ini          initial conformation for optimization
job.rsr          restraints file
job.sch          VTFM schedule file
job.B9999????? PDB atom file(s) for the model(s) of the target sequence
job.V9999????? violation profiles for the model(s)
job.D9999????? progress of optimization
job.BL9999????? optional loop model(s)
job.DL9999????? progress of optimization for loop model(s)
job.IL9999????? initial structures for loop model(s)

```

The main MODELLER routines used in each step are given in parentheses.

1. Read and check the alignment between the target sequence and the template structures (**READ_ALIGNMENT** and **CHECK_ALIGNMENT**).
2. Calculate restraints on the target from its alignment with the templates:
 - (a) Generate molecular topology for the target sequence (**GENERATE_TOPOLOGY**). Disulfides in the target are assigned here from the equivalent disulfides in the templates (**PATCH_SS_TEMPLATES**). Any user defined patches are also done here (as defined in TOP routine 'special_patches').
 - (b) Calculate coordinates for atoms that have equivalent atoms in the templates as an average over all templates (**TRANSFER_XYZ**) (alternatively, read the initial coordinates from a file).
 - (c) Build the remaining unknown coordinates using internal coordinates from the CHARMM topology library (**BUILD_MODEL**).
 - (d) Write the initial model to a file with extension `.ini` (**WRITE_MODEL**).
 - (e) Generate stereochemical, homology-derived, and special restraints (**MAKE_RESTRAINTS**) (alternatively, skip this and assume the restraints file already exists):

stereochemical	RESTRAINT_TYPE = 'bond angle dihedral improper'
mainchain dihedrals Φ, Ψ	RESTRAINT_TYPE = 'phi-psi_binormal'
mainchain dihedral ω	RESTRAINT_TYPE = 'omega_dihedral'
sidechain dihedral χ_1	RESTRAINT_TYPE = 'chi1_dihedral'
sidechain dihedral χ_2	RESTRAINT_TYPE = 'chi2_dihedral'
sidechain dihedral χ_3	RESTRAINT_TYPE = 'chi3_dihedral'
sidechain dihedral χ_4	RESTRAINT_TYPE = 'chi4_dihedral'
mainchain CA-CA distance	RESTRAINT_TYPE = 'distance'
mainchain N-O distance	RESTRAINT_TYPE = 'distance'
sidechain-mainchain distance	RESTRAINT_TYPE = 'distance'
sidechain-sidechain distance	RESTRAINT_TYPE = 'distance'
block distance restraints	RESTRAINT_TYPE = 'distance'
user defined	CALL ROUTINE = 'special_restraints'
non-bonded pairs distance	RESTRAINT_TYPE = 'sphere'; calculated on the fly
 - (f) Write all restraints to a file with extension `.rsr` (**WRITE_RESTRAINTS**).
3. Calculate model(s) that satisfy the restraints as well as possible. For each model:
 - (a) Generate the optimization schedule for the variable target function method (VTFM) (**MAKE_SCHEDULE**).
 - (b) Read the initial model (usually from the `.ini` file from 2.d) (**READ_MODEL**).
 - (c) Randomize the initial structure by adding a random number between \pm DEVIATION angstroms to all atomic positions (**RANDOMIZE_XYZ**).
 - (d) optimize the model:
 - Partially optimize the model by VTFM; Repeat the following steps as many times as specified by the optimization schedule:

- Read all the restraints by ‘rd_restraints’ (READ_RESTRAINTS).
- Select only the restraints that operate on the atoms that are close enough in sequence, as specified by the current step of VTFM (PICK_RESTRAINTS).
- Optimize the model by conjugate gradients, using only currently selected restraints (OPTIMIZE).
- Refine the model by simulated annealing with molecular dynamics, if so selected:
 - do a short conjugate gradients optimization (OPTIMIZE).
 - increase temperature in several steps and do molecular dynamics optimization at each temperature (OPTIMIZE).
 - decrease temperature in several steps and do molecular dynamics optimization at each temperature (OPTIMIZE).
 - do a short conjugate gradients optimization (OPTIMIZE).
- (e) calculate the remaining restraint violations and write them out (ENERGY).
- (f) write out the final model to a file with extension .B9999???? where ???? indicates the model number (WRITE_MODEL). Also write out the violations profile. Also write superposed templates and models if so selected by FINAL_MALIGN3D = 1.
- (g) superpose the models and the templates, if so selected, and write them out (EXPAND_ALIGNMENT, MALIGN3D).
- (h) do loop modeling if so selected (_loop).

3.2 Script for comparative modeling

The `_model` script implements the flowchart for comparative modeling by MODELLER that is described in the previous Section 3.1. The script uses routines in several other files. It is structured so that it is easy to deal with many different situations, some of which are described in Section 1.9. The script is too long to be listed here. It can be found in `$MODINSTALL6/bin/_model.top`. The default values of its arguments are defined in the `_defs` script file:

```
# Define additional TOP variables needed for MODELLER:

DEFINE_INTEGER VARIABLES = STARTING_MODEL ENDING_MODEL RSTRS_REFINED
DEFINE_INTEGER VARIABLES = LOOP_STARTING_MODEL LOOP_ENDING_MODEL
DEFINE_INTEGER VARIABLES = MAX_ITERATIONS_STORE WRITE_INTERMEDIATES
DEFINE_INTEGER VARIABLES = IREPEAT REPEAT_OPTIMIZATION EXIT_STAGE
DEFINE_INTEGER VARIABLES = CREATE_RESTRAINTS REFINE_HOT_ONLY
DEFINE_INTEGER VARIABLES = MAX_VAR_ITERATIONS FINAL_MALIGN3D INITIAL_MALIGN3D
DEFINE_INTEGER VARIABLES = DO_LOOPS ID1B
DEFINE_REAL VARIABLES = VIOL_REP_STORE MAX_MOLPDF
DEFINE_REAL VARIABLES = MAX_CA-CA_DISTANCE MAX_N-O_DISTANCE
DEFINE_REAL VARIABLES = MAX_SC-SC_DISTANCE MAX_SC-MC_DISTANCE
DEFINE_STRING VARIABLES = MODEL MODEL2 CODE CODE2 ALNFILE MODEL2_FIT
DEFINE_STRING VARIABLES = CSRFILE KNOWN SCHFILE FINAL_MODEL
DEFINE_STRING VARIABLES = GENERATE_METHOD RAND_METHOD MD_LEVEL
DEFINE_STRING VARIABLES = SEGFILE PDB_EXT TOPLIB PARLIB FAMILY FIT_IN_REFINE
DEFINE_STRING VARIABLES = ATOM_IDS1 ATOM_IDS2 OUTPUT2
DEFINE_STRING VARIABLES = LOOP_CSRFILE LOOP_INI_MODEL
DEFINE_STRING VARIABLES = LOOP_MD_LEVEL LOOP_INI_MODEL
DEFINE_STRING VARIABLES = LOOP_MODEL

# default values for options in comparative modeling by MODELLER:

SET STARTING_MODEL= 1 # the index of the first model;
```

```

# determines how many models are calculated;

SET ENDING_MODEL = 1 # the index of the last model;
# determines how many models are calculated;

SET DEVIATION = 4.0 # the amount of randomization of the initial model
# must be > 0 if different final models are wanted;

SET DO_LOOPS = '0' # whether or not to do automatic loop refinement
# for each model *.B???????? (0 for no, 1 for yes)
# can rely on automatic loop definition or
# re-define select_loop_atoms routine.
SET LOOP_STARTING_MODEL = 1 # how many loop models to generate for
SET LOOP_ENDING_MODEL = 25 # each *.B????????
SET LOOP_MD_LEVEL = 'refine_3' # the same as for MD_LEVEL, but for loops

#
# Do not forget to set WATER_IO, HETATM_IO, HYDROGEN_IO to ON if your model
# includes WATER, HYDROGEN, and/or HETATM atoms.
#
# Additional flexibility is provided by re-defining the TOP routines
# 'select_atoms', 'special_restraints', 'special_patches', and
# 'rd_restraints'.
#

# Options that are not changed frequently:

SET LIBRARY_SCHEDULE = 4 # 1 ... thorough var target func schedule
# 4 ... faster var target func schedule
SET MAX_VAR_ITERATIONS = 200 # maximal numb of iterations for the cycles
# of the variable target function method

SET MD_LEVEL = 'refine_3' # what kind of optimization is done after
# the variable target function method:
# 'nothing' ... nothing;
# 'refine_1' ... very fast MD annealing;
# 'refine_2' ... fast MD annealing;
# 'refine_3' ... slow MD annealing;
# 'refine_4' ... very slow MD annealing;
# 'refine_5' ... very slow/large dt MD annealing;

SET REFINE_HOT_ONLY = 0 # 1 ... select and optimize only HOT atoms in refine;
# 0 ... select and optimize all atoms in refine;
# usually about half of the atoms are hot; in such cases,
# 0 is faster for sequences longer than about 100 aa
# because a faster non-bonded pairs algorithm can be used.

SET RSTRS_REFINED = 1 # the types of restraints used to define
# hot spots when MD_LEVEL <> 'nothing':
# 0 ... stereochemistry only;
# 1 ... stereochemistry and dihedral;
# 2 ... all restraints;

SET EXIT_STAGE = 0 # 0 ... no effect;
# 1 ... exit without any optimization after
# restraints and an initial model are
# calculated (more efficient than
# REPEAT_OPTIMIZATION=0);

SET REPEAT_OPTIMIZATION = 1 # how many times the whole optimization
# schedule (variable target function

```



```

# method and refinement) is repeated
# for each initial model;

SET TRACE_OUTPUT = 10      # every which CG or MD cycle is reported;

SET MAX_MOLPDF = 100E3     # abort optimization of the current model if
# the molecular pdf is larger than this and
# continue with the next model;

SET TOPLIB = '${LIB}/top_heav.lib' # topology library (all non-hydrogen atoms);

SET TOPOLOGY_MODEL = 3     # corresponding topology model;

SET PARLIB = '${LIB}/par.lib'  # parameters library;

SET WRITE_INTERMEDIATES = 0   # 0 ... do not write out intermediate
# atom files during optimization;
# 1 ... write out intermediate atom files;

SET INITIAL_MALIGN3D = 0     # 0 ... do not do MALIGN3D before
# TRANSFER_XYZ
# 1 ... do that.

SET FINAL_MALIGN3D = 0      # 0 ... do not do MALIGN3D and write
# superposed templates & models
# at the end of 'model'
# 1 ... do that.

SET GENERATE_METHOD= 'transfer_xyz' # how to build the initial model:
# 'generate_xyz' from internal coordinates
# and write them to a file;
# 'transfer_xyz' from template coordinates
# and write them to a file;
# 'read_xyz' read coordinates from
# a file;

SET RAND_METHOD = 'randomize_xyz' # a method to perturb the initial model:
# 'randomize_dihedrals' ... uses DEVIATION
# in degrees;
# 'randomize_xyz' ... uses DEVIATION
# in angstroms;
# 'nothing'

SET CREATE_RESTRAINTS = 1    # 0 ... read the restraints from a file;
# 1 ... make the restraints and write them
# to a file before reading them
# for the optimization; in addition
# to the default restraints, the TOP
# routine 'special_restraints',
# which may be re-defined in the
# user TOP file, is called for any
# user defined restraints that are
# then also written to the same file.

SET SPLINE_ON_SITE = on     # on ... convert some restraints into splines
# off ... no conversion

# SET OUTPUT_CONTROL = 1 1 1 1 0 # write real_output, notes, warnings, errors, dynmem
SET OUTPUT_CONTROL = 1 0 0 1 0 # write real_output, notes, warnings, errors, dynmem

# Set maximal values for various distance restraints:
SET MAX_CA-CA_DISTANCE = 14.0

```

```

SET MAX_N-O_DISTANCE = 11.0
SET MAX_SC-MC_DISTANCE = 5.5
SET MAX_SC-SC_DISTANCE = 5.0

# Routine 'user_after_single_model' can be redefined to do whatever at the end
# of each model calculation (e.g. comparison with X-ray structure).

# To write out reports on individual optimizations:
SET OUTPUT = 'NO_REPORT SHORT'

# MSI for QUANTA:

# The alignment file format (I/O):
SET ALIGNMENT_FORMAT = 'PIR'

# The extension added to all *.Bxxxxnn filenames:
SET PDB_EXT = ' '
# SET PDB_EXT = '.modlr.pdb'

# to prevent SUPERPOSE in refine() if molecules are too small:
SET FIT_IN_REFINE = 'NO_FIT'

# To enable default filename generation if not explicitly defined:
SET MODEL = 'undefined'
SET CSRFILE = 'undefined'

# Call this routine before calling 'model' if you want real fast optimization
SUBROUTINE ROUTINE = 'very_fast'

# SET STARTING_MODEL = 1
# SET ENDING_MODEL = 1
SET MAX_CA-CA_DISTANCE = 10.0
SET MAX_N-O_DISTANCE = 6.0
SET MAX_SC-MC_DISTANCE = 5.0
SET MAX_SC-SC_DISTANCE = 4.5
# Note that all models will be the same if you do not change RAND_METHOD
SET RAND_METHOD = 'nothing'
SET MAX_VAR_ITERATIONS = 50
SET LIBRARY_SCHEDULE = 7
SET MD_LEVEL = 'nothing'

RETURN
END_SUBROUTINE

```

3.3 Script for modeling of loops

The new loop optimization method relies on a scoring function and optimization schedule adapted for loop modeling [?]. The corresponding TOP routine is called when you set DO_LOOPS to 1. Unless you re-define routine `select_loop_atoms`, all regions around gaps in the alignment, whether they are insertions or deletions, will be optimized thoroughly after the usual modeling procedure is finished. The loop optimization relies on a new scoring function adapted for loop modeling.

xx

Example:

```

# Homology modelling by the MODELLER TOP routine 'model'.
#
# This can be ran with run_clustor model-loop.top, too.

```

```
#
# In addition to the standard overall homology modeling, at the end, this
# routine also calls the thorough loop optimization routine, which generates
# by default 25 loop models for each *.B9999???? model. The default
# loop selection (regions around gaps) can be changed by re-defining
# routine select_loop_atoms.

INCLUDE                                # Include the predefined TOP routines

SET OUTPUT_CONTROL = 1 1 1 1 0

SET ALNFILE = 'alignment.ali'         # alignment filename
SET KNOWNNS = '5fd1'                  # codes of the templates
SET SEQUENCE = '1fdx'                 # code of the target
SET ATOM_FILES_DIRECTORY = './:../atom_files' # directories for input atom files
# SET STARTING_MODEL = 1
# SET ENDING_MODEL = 1
# (determines how many models to calculate)

SET DO_LOOPS = 1                      # do loops extensively
SET LOOP_STARTING_MODEL = 1
SET LOOP_ENDING_MODEL = 4
SET LOOP_MD_LEVEL = 'refine_1'
SET MD_LEVEL = 'nothing'

CALL ROUTINE = 'model'                # do homology modelling
```


Chapter 4

TOP, MODELLER scripting language

Chapter 5

Methods

5.1 Dynamic programming for sequence and structure comparison and searching

In this section, the basic dynamic programming method for sequence alignment is described [?]. This method forms the core of the pairwise and multiple sequence and structure comparisons as well as of the sequence database searching.

5.1.1 Pairwise comparison

The residue by residue scores W_{ij} can be used directly in the sequence alignment algorithm of Needleman & Wunsch [?] to obtain the comparison of two protein sequences or structures. The only difference between the two types of comparison is in the type of the comparison matrix. In the case of sequence, the amino acid substitution matrix is used. In the case of 3D structure, the Euclidean distance (or some function of it) between two equivalent atoms in the current optimal superposition is used [?].

The problem of the optimal alignment of two sequences as addressed by the algorithm of Needleman & Wunsch is as follows. We are given two sequences of elements and an M times N score matrix \mathcal{W} where M and N are the numbers of elements in the first and second sequence. The scoring matrix is composed of scores W_{ij} describing differences between elements i and j from the first and second sequence respectively. The goal is to obtain an optimal set of equivalences that match elements of the first sequence to the elements of the second sequence. The equivalence assignments are subject to the following “progression rule”: for elements i and k from the first sequence and elements j and l from the second sequence, if element i is equivalenced to element j , if element k is equivalenced to element l and if k is greater than i , l must also be greater than j . The optimal set of equivalences is the one with the smallest alignment score. The alignment score is a sum of scores corresponding to matched elements, also increased for occurrences of non-equivalenced elements (*ie* gaps). For a detailed discussion of this and related problems see [?].

We summarize the dynamic programming formulae used by MODELLER to obtain the optimal alignment since they differ slightly from those already published [?, ?]. The recursive dynamic programming formulae that give a matrix \mathcal{D} are:

$$\begin{aligned}
D_{i,j} &= \min \begin{cases} P_{i,j} \\ D_{i-1,j-1} + W_{i,j} \\ Q_{i,j} \end{cases} \\
P_{i,j} &= \min \begin{cases} D_{i-1,j} + g(1) \\ P_{i-1,j} + v \end{cases} \\
Q_{i,j} &= \min \begin{cases} D_{i,j-1} + g(1) \\ Q_{i,j-1} + v \end{cases}
\end{aligned} \tag{5.1}$$

where $g(l)$ is a linear gap penalty function:

$$g(l) = u + v \cdot l. \tag{5.2}$$

Note that only a vector is needed for the storage of P and Q . The uppermost formula in Eq. 5.1 is calculated for $i = M$ and $j = N$. Variable l is a gap length and parameters u and v are gap-penalty constants.

The arrays \mathcal{D} , \mathcal{P} and \mathcal{Q} are initialized as follows:

$$\begin{aligned}
D_{i,0} &= \begin{cases} 0, & i \leq e \\ g(i - e), & e < i \leq N \end{cases} \\
D_{0,j} &= \begin{cases} 0, & j \leq e \\ g(j - e), & e < j \leq N \end{cases} \\
P_{i,0} &= Q_{i,0} = \infty, & i = 1, 2, \dots, M \\
P_{0,j} &= Q_{0,j} = \infty, & j = 1, 2, \dots, N
\end{aligned} \tag{5.3}$$

where parameter e is the maximal number of elements at sequence termini which are not penalized with a gap-penalty if not equivalenced. A segment at the terminus of length e is termed an “overhang”. Note a difference from [?] in the initialization of the \mathcal{P} and \mathcal{Q} arrays. Also note that only vectors Q_i and P_j need to be stored in computer, not the whole arrays.

The minimal score $d_{M,N}$ is obtained from

$$d_{M,N} = \min(D_{i,N}, D_{M,j}) \tag{5.4}$$

where $i = M, M - 1, \dots, M - e$ and $j = N, N - 1, \dots, N - e$ to allow for the overhangs. The equivalence assignments are obtained by backtracking in matrix \mathcal{D} . Backtracking starts from the element $D_{i,j} = d_{M,N}$.

5.1.2 Variable gap penalty

This work is still in progress and is not described here.

5.1.3 Local *versus* global alignment

The Kruskal and Sankoff version of the local alignment is implemented [?]; this is very similar to the [?] method. All the routines for the local alignment are exactly the same as the routines for the global alignment except that during the construction of matrix D the alignment is restarted each time the score becomes higher than a cutoff. The second difference is that the backtracking starts from the lowest element in the matrix, wherever it is.

5.1.4 Similarity *versus* distance scores

Each scoring matrix contains a flag determining whether it is a distance or similarity matrix. An appropriate optimization is used automatically. This is achieved by using exactly the same code except that one side of comparisons is multiplied by -1 when dealing with similarities as opposed to distances.

5.1.5 Multiple comparisons

In the discussion of the previous section, we have assumed that the sequences or structures would be compared in a pairwise manner. However, such pairwise comparisons of several related proteins may not be self consistent, *ie* the following transitivity rule can be broken: If residue a from protein A is equivalent to residue b in protein B which in turn is equivalent to residue c in protein C then the residue a from protein A must also be equivalent to residue c from protein C . This property is not always attained in the set of usual pairwise comparisons relating a group of similar proteins. For this reason we proceed by simultaneously aligning all proteins. This is achieved by aligning the second sequence with the first one, the third sequence with the alignment of the first two, *etc.* A more general tree-like growth of the multiple alignment is not yet implemented.

If the number of all proteins is N , $N - 1$ alignments must be made to obtain the final multiple comparison. It is noted that once an equivalence or gap is introduced it is not changed in later stages.

5.2 Optimization of the objective function by MODELLER

This section describes the optimization methods implemented in MODELLER. The general form of the objective function and the structure of optimization are similar to molecular dynamics programs, such as CHARMM [?].

5.2.1 Function

MODELLER minimizes the *objective function* F with respect to Cartesian coordinates of $\sim 10,000$ atoms (3D points) that form a *system* (one or more molecules):

$$F = F(\mathbf{R}) = F_{symm} + \sum_i c_i(\mathbf{f}_i, \mathbf{p}_i) \quad (5.5)$$

where F_{symm} is an optional symmetry term defined in Eq. 5.71, \mathbf{R} are Cartesian coordinates of all atoms, c is a restraint i , \mathbf{f} is a geometric feature of a molecule, and \mathbf{p} are parameters. For a 10,000 atom system there can be on the order of 200,000 restraints. The form of c is simple; it includes a quadratic function, cosine, a weighted sum of a few Gaussian functions, Coulomb law, Lennard-Jones potential, cubic splines, and some other simple functions. The geometric features presently include a distance, an angle, a dihedral angle, a pair of dihedral angles between two, three, four atoms and eight atoms, respectively, the shortest distance in the set of distances (not documented further), solvent accessibility in \AA^2 , and atom density expressed as the number of atoms around the central atom. A pair of dihedral angles can be used to restrain such strongly correlated features as the mainchain dihedral angles Φ and Ψ . Each of the restraints also depends on a few parameters \mathbf{p}_i that generally vary from a restraint to a restraint. Some restraints can restrain *pseudo-atoms* such as a gravity center of several atoms.

MODELLER allows some atoms to be *fixed* during optimization; *i.e.*, only selected atoms are allowed to be moved. Similarly, MODELLER also allows only a subset of all restraints to be actually used in the calculation of the objective function. Each subset is indicated by a list of indices specifying the selected atoms or restraints.

There are two kinds of restraints, *static* and *dynamic*, that both contribute to the objective function as indicated in Eq. 5.5:

$$F = F_{symm} + F_s + F_d . \quad (5.6)$$

The static restraints and their parameters are pre-defined; *i.e.*, they are given before the call to the optimizer and are not changed during optimization. The dynamic restraints are re-generated repeatedly during optimization. Usually, the CPU time is spent evenly between the two kinds of restraints, although the dynamic restraints become more important as the size of the system increases. All dynamic restraints are always selected and they can restrain only pairs of atoms. In all other respects, the two kinds of restraints are the same.

The dynamic restraints are obtained from a *dynamic pairs list* (the non-bonded pairs list). Each dynamic pair corresponds to at least one restraint, which may or may not be violated. The dynamic pairs list includes only the pairs of atoms that satisfy the following three conditions: (1) One or both atoms in a pair are allowed to move. (2) The two atoms are not connected through one, two, or three chemical bonds. (3) The two atoms are closer than a preset cutoff distance (*e.g.*, 4 Å). There are on the order of 5000 atom pairs in the dynamic pairs list when only soft-sphere overlap restraints are used. Currently, the restraint types on the dynamic atom pairs that can be selected include the soft-sphere overlap, Lennard-Jones, Coulombic interactions, and MODELLER non-bonded spline restraints. xx atom density?

The existence of the dynamic pairs list is justified by the fact that dynamic pairs are usually a small fraction of all possible atom–atom pairs ($N \cdot (N - 1)/2$, where N is the number of atoms in a system). The use of the dynamic pairs list becomes especially beneficent as the size of the system increases.

The actual algorithm for creating the dynamic pairs list varies with the size of the system, whether or not all atoms are allowed to move, or whether or not the user wants to include the fixed environment in the calculation of non-bonded restraints involving the selected atoms. See Section 2.6.5 for more information.

The hash-function algorithm is used to determine whether or not two atoms are a dynamic atom pair. This algorithm is about 20 times slower than a lookup table but it requires much less memory and still spends a negligible fraction of the total CPU time. A hash-function table is prepared only once before the start of the optimization and any other operation involving an evaluation of the objective function (*e.g.*, **OPTIMIZE**, **ENERGY**, and **PICK_HOT_ATOMS**).

The dynamic pairs list is not necessarily re-generated each time the objective function is evaluated, although the contribution of the restraint to the objective function is calculated in each call to the objective function routine with the current values of the Cartesian coordinates. The dynamic pairs list is re-generated only when maximal atomic shifts accumulate to a value larger than a preset cutoff. This cutoff is chosen such that there cannot be a violation of a restraint without having its atom pair on the dynamic pairs list. The dynamic pairs list is recalculated in $\sim 20\%$ and $\sim 2\%$ of the objective function calls at the beginning and the end of optimization, respectively.

Each evaluation of the objective function or of its first derivatives with respect to the Cartesian coordinates involves the following steps:

1. Calculate non-fixed pseudo-atoms from the current atomic positions (routine `objfunc:pseudo`).
2. Update the dynamic pairs list, if necessary (routine `objfunc:upddyn`).
3. Calculate the violations of selected restraints and all other quantities that are shared between the calculations of the objective function and its derivatives (routine `objfunc:getviol`).
4. Sum the contributions of all violated restraints to the objective function and the derivatives (routine `objfunc:-getviol`).

5.2.2 Optimizers

MODELLER currently implements a Beale restart conjugate gradients algorithm [?, ?] and a molecular dynamics procedure with the Verlet integrator [?]. The conjugate gradients optimizer is usually used in combination with the variable target function method [?] which is implemented with the TOP script (Section 3.1). The molecular dynamics procedure can be used in a simulated annealing protocol that is also implemented with the TOP script.

Molecular dynamics

Force in MODELLER is obtained by equating the objective function F with internal energy in kcal/mole. The atomic masses are all set to that of C¹² (MODELLER unit is kg/mole). The initial velocities at a given temperature are obtained from a Gaussian random number generator with a mean and standard deviation of:

$$\bar{v}_x = 0 \quad (5.7)$$

$$\sigma_x = \sqrt{\frac{k_B T}{m}} = 0.000263143\sqrt{T} \quad (5.8)$$

where k_B is the Boltzmann constant, m is the mass of one C¹² atom, and the velocity is expressed in angstroms/-femtosecond.

The Newton's equations of motion are integrated by the Verlet algorithm [?]:

$$v_x(i+1) = v_x(i) + \frac{\partial F}{\partial x} A \quad (5.9)$$

$$x(i+1) = x(i) + v_x(i+1) \Delta t \quad (5.10)$$

$$A = c \frac{\Delta t}{m} = 4.1868 \cdot 10^{-7} \frac{\Delta t}{m} \quad (5.11)$$

where velocities $v(i+1)$ are for $t + \Delta t/2$ and positions $x(i+1)$ for $t + \Delta t$. Parameter c is a scaling factor so that positions are expressed in angstroms, time in femtoseconds, and velocities in angstroms/femtosecond, given that the objective function is in kcal/mole and atomic mass in kg/mole. In addition, velocity is capped at a maximum value, before calculating the shift, such that the maximal shift along one axis can only be CAP_ATOM_SHIFT. The velocities can be equilibrated every EQUILIBRATE steps to stabilize temperature. This is achieved by scaling the velocities with a factor f :

$$f = \sqrt{T/E_{kin}} \quad (5.12)$$

$$E_{kin} = \frac{m}{2} \sum_i^{N_{atoms}} (v_x^2 + v_y^2 + v_z^2) \quad (5.13)$$

where E_{kin} is the current kinetic energy of the system.

5.3 Equations used in the derivation of the molecular pdf

5.3.1 Features and their derivatives

Distance

Distance is defined by points i and j :

$$d = \sqrt{\vec{r}_{ij} \cdot \vec{r}_{ij}} = |\vec{r}_{ij}| = r_{ij} \quad (5.14)$$

where

$$\vec{r}_{ij} = \vec{r}_i - \vec{r}_j. \quad (5.15)$$

The first derivatives of d with respect to Cartesian coordinates are:

$$\frac{\partial d}{\partial \vec{r}_i} = \frac{\vec{r}_{ij}}{|\vec{r}_{ij}|} \quad (5.16)$$

$$\frac{\partial d}{\partial \vec{r}_j} = -\frac{\partial d}{\partial \vec{r}_i} \quad (5.17)$$

Angle

Angle is defined by points i , j , and k , and spanned by vectors ij and kj :

$$\alpha = \arccos \frac{\vec{r}_{ij} \cdot \vec{r}_{kj}}{|\vec{r}_{ij}| |\vec{r}_{kj}|}. \quad (5.18)$$

It lies in the interval from 0 to 180°. Internal MODELLER units are radians.

The first derivatives of α with respect to Cartesian coordinates are:

$$\frac{\partial \alpha}{\partial \vec{r}_i} = \frac{\partial \alpha}{\partial \cos \alpha} \frac{\partial \cos \alpha}{\partial \vec{r}_i} = \frac{1}{\sqrt{1 - \cos^2 \alpha}} \frac{1}{r_{ij}} \left(\frac{\vec{r}_{ij}}{r_{ij}} \cos \alpha - \frac{\vec{r}_{kj}}{r_{kj}} \right) \quad (5.19)$$

$$\frac{\partial \alpha}{\partial \vec{r}_k} = \frac{\partial \alpha}{\partial \cos \alpha} \frac{\partial \cos \alpha}{\partial \vec{r}_k} = \frac{1}{\sqrt{1 - \cos^2 \alpha}} \frac{1}{r_{kj}} \left(\frac{\vec{r}_{kj}}{r_{kj}} \cos \alpha - \frac{\vec{r}_{ij}}{r_{ij}} \right) \quad (5.20)$$

$$\frac{\partial \alpha}{\partial \vec{r}_j} = -\frac{\partial d}{\partial \vec{r}_i} - \frac{\partial d}{\partial \vec{r}_k} \quad (5.21)$$

These equations for the derivatives have a numerical instability when the angle goes to 0 or to 180°. Presently, the problem is ‘solved’ by testing for the size of the angle; if it is too small, the derivatives are set to 0 in the hope that other restraints will eventually pull the angle towards well behaved regions. Thus, angle restraints of 0 or 180° should not be used in the conjugate gradients or molecular dynamics optimizations.

Dihedral angle

Dihedral angle is defined by points i , j , k , and l ($ijkl$):

$$\chi = \text{sign}(\chi) \arccos \frac{(\vec{r}_{ij} \times \vec{r}_{kj}) \cdot (\vec{r}_{kj} \times \vec{r}_{kl})}{|\vec{r}_{ij} \times \vec{r}_{kj}| |\vec{r}_{kj} \times \vec{r}_{kl}|} \quad (5.22)$$

where

$$\text{sign}(\chi) = \text{sign}[\vec{r}_{kj} \cdot (\vec{r}_{ij} \times \vec{r}_{kj}) \times (\vec{r}_{kj} \times \vec{r}_{kl})]. \quad (5.23)$$

The first derivatives of χ with respect to Cartesian coordinates are:

$$\frac{d\chi}{d\vec{r}} = \frac{d\chi}{d \cos \chi} \frac{d \cos \chi}{d\vec{r}} \quad (5.24)$$

where

$$\frac{d\chi}{d \cos \chi} = \left(\frac{d \cos \chi}{d\chi} \right)^{-1} = -\frac{1}{\sin \chi} \quad (5.25)$$

and

$$\frac{\partial \cos \chi}{\partial \vec{r}_i} = \vec{r}_{kj} \times \vec{a} \quad (5.26)$$

$$\frac{\partial \cos \chi}{\partial \vec{r}_j} = \vec{r}_{ik} \times \vec{a} - \vec{r}_{kl} \times \vec{b} \quad (5.27)$$

$$\frac{\partial \cos \chi}{\partial \vec{r}_k} = \vec{r}_{jl} \times \vec{b} - \vec{r}_{ij} \times \vec{a} \quad (5.28)$$

$$\frac{\partial \cos \chi}{\partial \vec{r}_l} = \vec{r}_{ij} \times \vec{b} \quad (5.29)$$

$$\vec{a} = \frac{1}{|\vec{r}_{ij} \times \vec{r}_{kj}|} \left(\frac{\vec{r}_{kj} \times \vec{r}_{kl}}{|\vec{r}_{kj} \times \vec{r}_{kl}|} - \cos \chi \frac{\vec{r}_{ij} \times \vec{r}_{kj}}{|\vec{r}_{ij} \times \vec{r}_{kj}|} \right) \quad (5.30)$$

$$\vec{b} = \frac{1}{|\vec{r}_{kj} \times \vec{r}_{kl}|} \left(\frac{\vec{r}_{ij} \times \vec{r}_{kj}}{|\vec{r}_{ij} \times \vec{r}_{kj}|} - \cos \chi \frac{\vec{r}_{kj} \times \vec{r}_{kl}}{|\vec{r}_{kj} \times \vec{r}_{kl}|} \right). \quad (5.31)$$

These equations for the derivatives have a numerical instability when the angle goes to 0. Thus, the following set of equations is used instead [?]:

$$\vec{r}_{mj} = \vec{r}_{ij} \times \vec{r}_{kj} \quad (5.32)$$

$$\vec{r}_{nk} = \vec{r}_{kj} \times \vec{r}_{kl} \quad (5.33)$$

$$\frac{\partial \chi}{\partial \vec{r}_i} = \frac{r_{kj}}{r_{mj}^2} \vec{r}_{mj} \quad (5.34)$$

$$\frac{\partial \chi}{\partial \vec{r}_i} = -\frac{r_{kj}}{r_{nk}^2} \vec{r}_{nk} \quad (5.35)$$

$$\frac{\partial \chi}{\partial \vec{r}_j} = \left(\frac{\vec{r}_{ij} \cdot \vec{r}_{kj}}{r_{kj}^2} - 1 \right) \frac{\partial \chi}{\partial \vec{r}_i} - \frac{\vec{r}_{kl} \cdot \vec{r}_{kj}}{r_{kj}^2} \frac{\partial \chi}{\partial \vec{r}_i} \quad (5.36)$$

$$\frac{\partial \chi}{\partial \vec{r}_k} = \left(\frac{\vec{r}_{kl} \cdot \vec{r}_{kj}}{r_{kj}^2} - 1 \right) \frac{\partial \chi}{\partial \vec{r}_i} - \frac{\vec{r}_{ij} \cdot \vec{r}_{kj}}{r_{kj}^2} \frac{\partial \chi}{\partial \vec{r}_i} \quad (5.37)$$

The only possible instability in these equations is when the length of the central bond of the dihedral, r_{kj} , goes to 0. In such a case, which should not happen, the derivatives are set to 0. The expressions for an improper dihedral angle, as opposed to a dihedral or dihedral angle, are the same, except that indices $ijkl$ are permuted to $ikjl$. In both cases, covalent bonds ij , jk , and kl are defining the angle.

Atomic solvent accessibility

xx

Atomic density

xx

5.3.2 Restraints and their derivatives

The chain rule is used to find the partial derivatives of the feature pdf with respect to the atomic coordinates. Thus, only the derivatives of the pdf with respect to the features are listed here.

Single Gaussian restraint

The pdf for a geometric feature f (e.g., distance, angle, dihedral angle) is

$$p = \frac{1}{\sigma\sqrt{2\pi}} \exp \left[-\frac{1}{2} \left(\frac{f - \bar{f}}{\sigma} \right)^2 \right]. \quad (5.38)$$

A corresponding restraint c in the sum that defines the objective function F is

$$c = -\ln p = \frac{1}{2} \left(\frac{f - \bar{f}}{\sigma} \right)^2 - \ln \frac{1}{\sigma\sqrt{2\pi}} \quad (5.39)$$

The first derivatives with respect to feature f are:

$$\frac{dc}{df} = \frac{f - \bar{f}}{\sigma} \frac{1}{\sigma}. \quad (5.40)$$

Multiple Gaussian restraint

The polymodal pdf for a geometric feature f (e.g., distance, angle, dihedral angle) is

$$p = \sum_{i=1}^n \omega_i p_i = \sum_{i=1}^n \omega_i \frac{1}{\sigma_i\sqrt{2\pi}} \exp \left[-\frac{1}{2} \left(\frac{f - \bar{f}_i}{\sigma_i} \right)^2 \right]. \quad (5.41)$$

A corresponding restraint c in the sum that defines the objective function F is

$$c = -\ln p = -\ln \sum_{i=1}^n \omega_i p_i \quad (5.42)$$

The first derivatives with respect to feature f are:

$$\frac{dc}{df} = \frac{1}{p} \sum_{i=1}^n \omega_i p_i \cdot \left[\frac{f - \bar{f}_i}{\sigma_i} \frac{1}{\sigma_i} \right]. \quad (5.43)$$

When any of the normalized deviations $v_i = (f - \bar{f}_i)/\sigma_i$ is large, there are numerical instabilities in calculating the derivatives because v_i are arguments to the *exp* function. Robustness is ensured as follows. The ‘effective’ normalized deviation is used in all the equations above when the magnitude of normalized violation v is larger than cutoff `rgauss1` (10 for double precision). This scheme works up to `rgauss2` (200 for double precision); violations larger than that are ignored. This trick is equivalent to increasing the standard deviation σ_i . A slight disadvantage is that there is a discontinuity in the first derivatives at `rgauss1`. However, if continuity were imposed, the range would not be extended (this is equivalent to linearizing the Gaussian, but since it is already linear for large deviations, a linearization with derivatives smoothness would not introduce much change at all).

$$M = 37 \quad ; \quad M^2/2 \text{ has to be smaller than the largest argument to exp} \quad (5.44)$$

$$A = \frac{1}{M} \frac{\text{rgauss2} - M}{\text{rgauss2} - \text{rgauss1}} \quad (5.45)$$

$$B = \frac{\text{rgauss2}}{M} \frac{M - \text{rgauss1}}{\text{rgauss2} - \text{rgauss1}} \quad (5.46)$$

$$v = \frac{f - \bar{f}_i}{\sigma_i} \quad (5.47)$$

$$F = A|v| + B \quad (5.48)$$

$$v' = v/F \quad (5.49)$$

Now, Eqs. 5.41–5.43 are used with v' instead of v . For single precision, $M = 12$, `rgauss1` = 4, `rgauss2` = 100.

Multiple binormal restraint

The polymodal pdf for a geometric feature (f_1, f_2) (*e.g.*, a pair of dihedral angles) is

$$p = \sum_{i=1}^n \omega_i p_i = \sum_{i=1}^n \omega_i \frac{1}{2\pi\sigma_{1i}\sigma_{2i}\sqrt{(1-\rho_i^2)}} \cdot \exp \left\{ -\frac{1}{2(1-\rho_i^2)} \left[\left(\frac{f_1 - \bar{f}_{1i}}{\sigma_{1i}} \right)^2 - 2\rho_i \frac{f_1 - \bar{f}_{1i}}{\sigma_{1i}} \frac{f_2 - \bar{f}_{2i}}{\sigma_{2i}} + \left(\frac{f_2 - \bar{f}_{2i}}{\sigma_{2i}} \right)^2 \right] \right\}. \quad (5.50)$$

where $\rho < 1$. ρ is the correlation coefficient between f_1 and f_2 . A corresponding restraint c in the sum that defines the objective function F is

$$c = -\ln p = -\ln \sum_{i=1}^n \omega_i p_i \quad (5.51)$$

The first derivatives with respect to features f_1 and f_2 are:

$$\frac{\partial c}{\partial f_1} = \frac{1}{p} \sum_{i=1}^n \left[\omega_i p_i \cdot \frac{1}{\sigma_{1i}(1-\rho_i^2)} \left(\frac{f_1 - \bar{f}_{1i}}{\sigma_{1i}} - \rho_i \frac{f_2 - \bar{f}_{2i}}{\sigma_{2i}} \right) \right] \quad (5.52)$$

$$\frac{\partial c}{\partial f_2} = \frac{1}{p} \sum_{i=1}^n \left[\omega_i p_i \cdot \frac{1}{\sigma_{2i}(1-\rho_i^2)} \left(\frac{f_2 - \bar{f}_{2i}}{\sigma_{2i}} - \rho_i \frac{f_1 - \bar{f}_{1i}}{\sigma_{1i}} \right) \right]. \quad (5.53)$$

Lower bound

This is like the left half of a single Gaussian restraint:

$$p = \begin{cases} p_{gauss} & ; \quad f < \bar{f} \\ 0 & ; \quad f \geq \bar{f} \end{cases} \quad (5.54)$$

where \bar{f} is a lower bound and p_{gauss} is given in Eq. 5.38. A similar equation relying on the first derivatives of a Gaussian p holds for the first derivatives of a lower bound.

Upper bound

This is like the right half of a single Gaussian restraint:

$$p = \begin{cases} p_{gauss} & ; \quad f > \bar{f} \\ 0 & ; \quad f \leq \bar{f} \end{cases} \quad (5.55)$$

where \bar{f} is an upper bound and p_{gauss} is given in Eq. 5.38. A similar equation relying on the first derivatives of a Gaussian p holds for the first derivatives of an upper bound.

Cosine restraint

This is usually used for dihedral angles f :

$$c = |b| - b \cos(nf + a) \quad (5.56)$$

where b is CHARMM force constant, a is phase shift (tested for 0 and 180°), and n is periodicity (tested for 1, 2, 3, 4, 5, and 6). The CHARMM phase value from the CHARMM parameter library corresponds to $a - 180^\circ$. The force constant b can be negative, in effect offsetting the phase a for 180° compared to the same but positive force constant.

$$\frac{dc}{df} = bn \sin(nf + a) \quad (5.57)$$

Coulomb restraint

$$c = \frac{q_i q_j}{f} s(f, f_1, f_2) \quad (5.58)$$

$$s(f, f_1, f_2) = \begin{cases} 1 & ; \quad f \leq f_1 \\ \frac{(f_2 - f)^2 (f_2 + 2f - 3f_1)}{(f_2 - f_1)^3} & ; \quad f_1 < f \leq f_2 \\ 0 & ; \quad f > f_2 \end{cases} \quad (5.59)$$

where q_i and q_j are the atomic charges of atoms i and j , obtained from the CHARMM topology file, that are at a distance f . Function $s(f, f_1, f_2)$ is a switching function that smoothes the potential down to zero in the interval from f_1 to f_2 ($f_2 > f_1$). The total Coulomb energy of a molecule is a sum over all pairs of atoms that are not in the same bonds or bond angles. 1-4 energy for the 1-4 atom pairs in the same dihedral angle corresponds to the ELEC14 MODELLER term; the remaining longer-range contribution corresponds to the ELEC term.

The first derivatives are:

$$\frac{dc}{df} = -\frac{c}{f} + \frac{c}{s} \frac{ds}{df} \quad (5.60)$$

$$\frac{ds}{df} = \begin{cases} 0 & ; \quad f \leq f_1 \\ \frac{6(f_2 - f)(f_1 - f)}{(f_2 - f_1)^3} & ; \quad f_1 < f \leq f_2 \\ 0 & ; \quad f > f_2 \end{cases} \quad (5.61)$$

Lennard-Jones restraint

Usually used for non-bonded distances:

$$c = \left[\left(\frac{A}{f} \right)^{12} - \left(\frac{B}{f} \right)^6 \right] s(f, f_1, f_2) \quad (5.62)$$

The parameters f_1 and f_2 of the switching function can be different from those in Eq. 5.59. The parameters A and B are obtained from the CHARMM parameter file (NONBOND section) where they are given as E_i and r_j such that $E_{ij}(f) = -4\sqrt{E_i E_j}[(\rho_{ij}/f)^{12} - (\rho_{ij}/f)^6]$ in kcal/mole for f in angstroms and $\rho = (r_i + r_j)/2^{1/6}$; the minimum of E is $-\sqrt{E_i E_j}$ at $f = (r_i + r_j)$, and its zero is at $f = \rho$. The total Lennard-Jones energy should be evaluated over all pairs of atoms that are not in the same bonds or bond angles. The parameters A and B for 1–4 pairs in dihedral angles can be different from those for the other pairs; they are obtained from the second set of E_i and r_i in the CHARMM parameter file, if it exists. 1–4 energy corresponds to the LJ14 MODELLER term; the remaining longer-range contribution corresponds to the LJ term.

The first derivatives are:

$$\frac{dc}{df} = \frac{Cs}{f} - C \frac{ds}{df} \quad (5.63)$$

$$C = -12 \left(\frac{A}{f} \right)^{12} + 6 \left(\frac{B}{f} \right)^6 \quad (5.64)$$

Spline restraint

Any restraint form can be represented by a cubic spline [?]:

$$c = Ac_j + Bc_{j+1} + Cc'_j + Dc'_{j+1} \quad (5.65)$$

$$A = \frac{f_{j+1} - f}{f_{j+1} - f_j} \quad (5.66)$$

$$B = 1 - A \quad (5.67)$$

$$C = \frac{1}{6}(A^3 - A)(f_{j+1} - f_j)^2 \quad (5.68)$$

$$D = \frac{1}{6}(B^3 - B)(f_{j+1} - f_j)^2 \quad (5.69)$$

where $f_j \leq f \leq f_{j+1}$.

The first derivatives are:

$$\frac{dc}{df} = \frac{c_{j+1} - c_j}{f_{j+1} - f_j} - \frac{3A^2 - 1}{6}(f_{j+1} - f_j)c'_j + \frac{3B^2 - 1}{6}(f_{j+1} - f_j)c'_{j+1} \quad (5.70)$$

The values of c and c' beyond f_1 and f_n are obtained by linear interpolation from the termini. A violation of the restraint is calculated by finding the global minimum. A relative violation is estimated by using a standard deviation (*e.g.*, force constant) obtained by fitting a parabola to the global minimum.

Variable spacing of spline points could be used to save on memory. However, this would increase the execution time, so it is not used.

Symmetry restraint

The asymmetry penalty added to the objective function is defined as

$$F_{symm} = \sum_{i < j} \omega_i \omega_j (d_{ij} - d'_{ij})^2 \quad (5.71)$$

where the sum runs over all pairs of equivalent atoms ij , ω_i is an atom weight for atom i , d_{ij} is an intra-molecular distance between atoms ij in the first segment, and d'_{ij} is the equivalent distance in the second segment.

For each $i < j$, the first derivatives are:

$$\frac{\partial c}{\partial \vec{d}_{ij}} = 2\omega_i\omega_j(d_{ij} - d'_{ij})\frac{\vec{d}_{ij}}{d_{ij}} \quad (5.72)$$

$$\frac{\partial c}{\partial \vec{d}'_{ij}} = -2\omega_i\omega_j(d_{ij} - d'_{ij})\frac{\vec{d}'_{ij}}{d'_{ij}} \quad (5.73)$$

Thus, the total first derivatives are obtained by summing the two expressions above for all i and $j > i$ distances.

5.4 List of commands, arguments, and default values

The top.ini file contains the list of all MODELLER commands, arguments, and default values of arguments.

```

--- COMMANDS:
 1 no_action
 2 SET
 3 STOP
 4 LABEL
 5 GO_TO
 6 DEFINE_INTEGER
 7 DEFINE_REAL
 8 END_DO
 9 DO
10 CALL
11 RESET
12 WRITE
13 OPERATE
14 STRING_OPERATE
15 DEFINE_STRING
16 DEFINE_LOGICAL
17 SUBROUTINE
18 END_SUBROUTINE
19 INCLUDE
20 RETURN
21 READ
22 OPEN
23 CLOSE
24 IF
25 WRITE_TOP
26 SYSTEM
27 INQUIRE
28 STRING_IF
31 READ_RESTRAINTS
32 READ_SCHEDULE
33 WRITE_RESTRAINTS
34 READ_MODEL
35 SUPERPOSE
36 COMPARE
37 WRITE_MODEL
38 WRITE_MODEL2
39 OPTIMIZE
40 ENERGY
41 READ_MODEL2
42 PICK_ATOMS
43 ROTATE_DIHEDRALS
44 READ_ALIGNMENT
45 DELETE_ALIGNMENT
46 SWITCH_TRACE

```

```
47 PATCH
48 TRANSFER_RES_NUMB
49 MAKE_SCHEDULE
50 WRITE_SCHEDULE
51 ID_TABLE
52 MAKE_WRITE_UHBD_LIB
53 BUILD_MODEL
54 GENERATE_TOPOLOGY
55 MAKE_RESTRAINTS
56 READ_TOPOLOGY
57 READ_PARAMETERS
58 WRITE_TOPOLOGY_MODEL
59 MAKE_TOPOLOGY_MODEL
60 ROTATE_MODEL
61 WRITE_ALIGNMENT
62 REORDER_ATOMS
63 PICK_RESTRAINTS
64 CONDENSE_RESTRAINTS
65 DELETE_RESTRAINT
66 ADD_RESTRAINT
67 TRANSFER_XYZ
68 RANDOMIZE_XYZ
69 DEBUG_FUNCTION
70 undefined70
71 PICK_HOT_ATOMS
72 REINDEX_RESTRAINTS
73 ALIGN
74 SEQUENCE_SEARCH
75 ALIGN3D
76 ORIENT_MODEL
77 DESCRIBE
78 SEQUENCE_COMPARISON
79 MALIGN3D
80 MALIGN
81 SEQUENCE_TO_ALI
82 PMF
83 MUTATE_MODEL
84 PATCH_SS_MODEL
85 WRITE_DATA
86 PRINCIPAL_COMPONENTS
87 READ_ALIGNMENT2
88 COMPARE_ALIGNMENTS
89 ALIGN_CONSENSUS
90 QUICK_AND_DIRTY
91 SPLINE_RESTRAINTS
92 RENAME_SEGMENTS
93 DEFINE_SYMMETRY
94 PATCH_SS_TEMPLATES
95 CHECK_ALIGNMENT
96 ALIGN2D
97 COLOR_ALN_MODEL
98 IUPAC_MODEL
99 DENDROGRAM
100 EXPAND_ALIGNMENT
101 UNBUILD_MODEL
102 READ_ATOM_CLASSES
103 SEGMENT_MATCHING
104 READ_RESTYP_LIB
105 WRITE_PDB_XREF
106 MAKE_REGION
--- KEYWORDS:
```

1	REAL	ARGUMENTS	0 0.00	# real arguments to the math operation
31	REAL	UPDATE_DYNAMIC	1 0.39	# when to update non-bonded pairs list
32	REAL	MATRIX_OFFSET	1 0.00	# substitution matrix offset for local alignment
33	REAL	SPHERE_STDV	1 0.05	# standard deviation of soft-sphere repulsion
34	REAL	VIOL_REPORT_CUT	33 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 4.5 999 999 999 999 4.5 4.5 4.5	
35	REAL	DEBUG_FUNCTION_CUTOFF	3 0.01 0.001 0.1	# cutoffs for reporting differences between numerical and analytical
36	REAL	TRANSLATION	3 0.0 0.0 0.0	# translation vector for MODEL
37	REAL	SA_STEP	1 0.2	# amplitude of the Monte Carlo steps
38	REAL	SA_MVFRACT	1 0.8	# fraction of accepted Monte Carlo steps
39	REAL	SA_TFACTR	1 0.9	# factor for temperature decrease in MC SA
40	REAL	SA_TO	1 40.0	# starting SA temperature
41	REAL	SA_TMIN	1 0.01	# final SA temperature
42	REAL	MIN_ATOM_SHIFT	1 0.010	# minimal atomic shift for the optimization convergence test
43	REAL	DEVIATION	1 0.0	# coordinate randomization amplitude in angstroms
44	REAL	RMS_CUTOFFS	11 3.5 3.5 60 60 15 60 60 60 60 60 60	# cutoffs for RMS, DRMS, Alpha Phi Psi Omega
45	REAL	TEMPERATURE	1 293.0	# temperature for MD simulation in K
46	REAL	MD_TIME_STEP	1 4.0	# time step for MD in fs
47	REAL	RADII_FACTOR	1 0.82	# factor for van der Waals radii
48	REAL	LENNARD_JONES_SWITCH	2 6.5 7.5	# the range for Lennard-Jones interaction smoothing to 0
49	REAL	COULOMB_SWITCH	2 6.5 7.5	# the range for Coulomb interaction smoothing to 0
50	REAL	ROTATION_MATRIX	9 1 0 0 0 1 0 0 0 1	# rotation matrix for MODEL
51	REAL	BASIS_RELATIVE_WEIGHT	1 0.05	# the cutoff weight of basis pdf's for their removal
52	REAL	SYMMETRY_WEIGHT	1 1.0	# the weight of the symmetry objective function term
53	REAL	MAXIMAL_DISTANCE	1 999	# maximal distance for distance restraints
54	REAL	RESTRAINTS_FILTER	33 -999 -999 -999 -999 -999 -999 -999 -999 -999 -999 -999 -999 -999 -999 -999	
55	REAL	RESTRAINT_PARAMETERS	0 3 1 3 3 4 2 0 0.0 0.087	# restraint parameters
56	REAL	SPHERE_RADIUS	1 10.0	# sphere radius for atoms selection
57	REAL	SELECTION_SLAB	5 -9999 9999 0 0 0	# slab for atoms selection: \Z{dz1} \Z{dz2} \Z{xtrans} \Z{ytran
58	REAL	PICK_HOT_CUTOFF	1 4.0	# radius for picking hot atoms
59	REAL	CAP_ATOM_SHIFT	1 0.2	# limit for atomic shifts in optimization
60	REAL	MOLPDF	1 0.0	# value of objective function
61	REAL	GAP_PENALTIES_3D	2 0.0 1.75	# gap creation and extension penalties for structure/structure superpos
62	REAL	CONTACT_SHELL	1 4.0	# distance cutoff for calculation of the non-bonded pairs list
63	REAL	RESTRAINT_STDEV	2 0.0 1.0	# transforming factors for standard deviations (y=a+bx) in models 1--6 o
64	REAL	PMF_GRID	8 2.0 0.5 20 36 18 0 180 1	# translation and rotation grid for PMF calculation
65	REAL	RELATIVE_DIELECTRIC	1 1.0	# relative dielectric constant
66	REAL	ROTATION_ANGLE	1 0.0	# rotation of MODEL around axis [degrees]
67	REAL	ROTATION_AXIS	3 1.0 0.0 0.0	# rotation axis for MODEL
68	REAL	SPLINE_DX	1 0.5	# interval size for splining restraints
69	REAL	SPLINE_RANGE	1 4.0	# range of the splines
70	REAL	GAP_PENALTIES_2D	8 0.35 1.2 0.9 1.2 0.6 8.6 1.2 0.0	# gap penalties for sequence/structure alignment
71	REAL	SCHEDULE_SCALE	33 1	# factors for
72	REAL	CLUSTER_CUT	1 -1.0	# definition of a cluster
73	REAL	GAP_PENALTIES_1D	2 -900 -50	# gap creation and extension penalties for sequence/sequence alignment
74	REAL	FAST_SEARCH_CUTOFF	1 1.0	# if FAST_SEARCH is ON only sequences with database scan significance higher
75	REAL	VIOL_REPORT_CUT2	33 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0	
76	REAL	SIGNIF_CUTOFF	2 4.0 5.0	# cutoff for adding sequences to alignment, max difference from the best
77	REAL	SEGMENT_CUTOFF	1 9999999	# cutoff for writing out an alignment in SEGMENT_MATCHING
78	REAL	FIX_OFFSETS	5 0. 1000. 2000. 3000. 4000.	# offsets of the ALIGN2D alignment score for "fixed"
79	REAL	PSA_INTEGRATION_STEP	1 0.1	# integration step for WRITE_DATA
80	REAL	ATOM_ACCESSIBILITY	1 1.0	# accessible atoms for MAKE_REGION
81	REAL	PROBE_RADIUS	1 1.4	# probe_radius for WRITE_DATA
82	REAL	REFERENCE_DISTANCE	1 3.5	# cutoff for selecting reference positions in SUPERPOSE
1	INTEGER	IO_UNIT	1 21	# IO unit for file operations
2	INTEGER	ID1	1 1	# ID1 for filename construction
3	INTEGER	ID2	1 1	# ID2 for filename construction
4	INTEGER	NUMBER_PLACES	2 5 2	# pre- and post-decimal point places
5	INTEGER	FILE_EXISTS	1 0	# an output flag: 0 1
6	INTEGER	OUTPUT_CONTROL	5 1 0 1 1 0	# selects output, flow-control msgs, warnings, errors, dynamic mem msg
7	INTEGER	STOP_ON_ERROR	1 1	# whether to stop on error
8	INTEGER	ERROR_STATUS	1 0	# application error status

```

 9 INTEGER NUMBER_LINES          1 0 # number of lines in the newly opened file
31 INTEGER SCHEDULE_STEP        1 1 # schedule step for optimization
32 INTEGER ROUTINE_TYPE         1 1 # generic routine type for a miscellaneous command
33 INTEGER NLOGN_USE            1 15 # number of residues at which to begin using the N Log N non-bonded pairs rou
34 INTEGER SA_MOVSPEARM        1 30 #
35 INTEGER RESIDUE_GROUPING     1 1 #
36 INTEGER MAX_ITERATIONS       1 200 # maximal iterations in optimization
37 INTEGER RAND_SEED            1 -8123 # random seed from -50000 to -2
38 INTEGER COMPARE_MODE         1 3 # selects the type of comparison: 1 | 2 | 3
39 INTEGER EXTEND_HOT_SPOT      1 0 # whether to extend hot spots
40 INTEGER TOPOLOGY_MODEL       1 3 # selects topology library: 1--9
41 INTEGER RENUMBER_RESIDUES    0 # starting residue index for renumbering residues
42 INTEGER N_SCHEDULE           1 1 # the number of steps in the optimization schedule
43 INTEGER DISTANCE_RSR_MODEL   1 1 # the model for calculating distance restraints: 1--7
44 INTEGER ACCESSIBILITY_TYPE   1 8 # type of solvent accessibility: 1--10
45 INTEGER RESIDUE_SPAN_RANGE   2 0 99999 # range of residues spanning the allowed distances; for MAKE_RESTRAINTS,
46 INTEGER MAX_GAP_LENGTH       1 999999 # maximal length of gap in protein comparisons
47 INTEGER OPTIMIZATION_METHOD  1 -999 # type of optimization method: 1 | 3
48 INTEGER GAP_EXTENSION        2 2 1 # extend insertions/deletions for that many residues, in PICK_ATOMS; don't s
49 INTEGER NUMB_OF_SEQUENCES    1 1 # number of sequences in the alignment
50 INTEGER TRACE_OUTPUT         1 0 # modulus for writing information about optimization iterations: 0 for nothing
51 INTEGER SEARCH_TOP_LIST      1 20 # the length of the output hits list
52 INTEGER EQUILIBRATE          1 999999 # equilibrate during MD every that many steps
53 INTEGER MAX_GAPS_MATCH       1 1 #
54 INTEGER ALIGN_BLOCK          1 1 # the last sequence in the first block of sequences
55 INTEGER PICK_ATOMS_SET       1 1 # index of the selected atoms set: 1 | 2 | 3
56 INTEGER PMF_INDICES          0 0 0 0 0 #
57 INTEGER SEARCH_RANDOMIZATIONS 1 0 # number of randomizations for calculating the significance of a sequence/sequ
58 INTEGER OFF_DIAGONAL         1 100 # to speed up the alignment
59 INTEGER RESTRAINT_GROUP      1 26 # physical restraint group
60 INTEGER OVERHANG             1 0 # un-penalized overhangs in protein comparisons
61 INTEGER SPLINE_SELECT        3 4 1 9 # specification of the restraints to be splined: {\tt form feature group
62 INTEGER LIBRARY_SCHEDULE     1 1 # selects schedule from the $SCHED_LIB library
63 INTEGER NONBONDED_SEL_ATOMS  1 1 # a non-bonded pair has to have at least as many selected atoms
64 INTEGER SPLINE_MIN_POINTS    1 5 # have at least as many intervals in a spline
65 INTEGER SHEET_H-BONDS        1 7 # specify hydrogen bonds in a beta-sheet
66 INTEGER SMOOTHING_WINDOW     1 3 # profiles are smoothed over 2*SW + 1 residues
67 INTEGER RESTRAINT_SEL_ATOMS  1 1 # a restraint other than non-bonded pair has to have at least as many selected
68 INTEGER undefined1           1 0 #
69 INTEGER PROFILE_2D_PHYS      1 33 # 1 ... 33 physical type to be presented as 2D energy profile2
70 INTEGER MIN_LOOP_LENGTH      0 # inter-segment minimal lengths in SEGMENT_MATCHING
71 INTEGER SEGMENT_SHIFTS       0 # segment shifts +- in SEGMENT_MATCHING
72 INTEGER SEGMENT_REPORT       1 1D6 # for SEGMENT_MATCHING
73 INTEGER MNCH_LIB             1 1 # which MNCH lib to use in MAKE_RESTRAINTS
74 INTEGER SEGMENT_GROWTH_N     0 # reducing/growing segment N-termini in SEGMENT_MATCHING
75 INTEGER SEGMENT_GROWTH_C     0 # reducing/growing segment C-termini in SEGMENT_MATCHING
76 INTEGER EXPAND_CONTROL       5 9999 9999 1 10 0 # for controlling EXPAND_ALIGNMENT
77 INTEGER NUMB_OF_SEQUENCES2   1 0 # number of sequence in ALIGNMENT2
78 INTEGER MAXRES               0 0 # user specified maximal number of residues
79 INTEGER REGION_SIZE          1 20 # size of exposed region in MAKE_REGION
80 INTEGER MAX_LOOP_LENGTH      1 15 # maximal length of a loop in PICK_ATOMS
 1 STRING OBJECTS               0 '' # variable names or constants
 2 STRING VARIABLES             0 '' # variable names
 3 STRING ROUTINE               0 '' # subroutine name
 4 STRING ROOT_NAME             1 'undf' # root of a filename for filename construction
 5 STRING DIRECTORY             1 '' # directory list (e.g., \Z{dir1:dir2:dir3.:/:})
 6 STRING FILE_ID               1 'default' # file id for filename construction
 7 STRING OPERATION             1 'SUM' # operation to perform: \Z{SUM} | \Z{MULTIPLY} | \Z{DIVIDE} | \Z{POWER} |
 8 STRING RESULT                0 '' # variable name for the result of operation
 9 STRING STRING_ARGUMENTS      0 '' # arguments for string operation
10 STRING OBJECTS_FILE          1 'top.out' # filename

```

```

11 STRING INCLUDE_FILE      1 '_mod' # include file name
12 STRING FILE              1 'default' # partial or complete filename
13 STRING RECORD            1 'undefined' # contents of the input line
14 STRING THEN              1 'undefined' # statement when IF evaluates to T
15 STRING ELSE              1 'undefined' # statement when IF evaluates to F
16 STRING COMMAND           1 'nothing' # UNIX command
17 STRING FILE_EXT          1 '' # file extension for filename construction
18 STRING OUTPUT_DIRECTORY 1 '' # output directory
19 STRING FILE_ACCESS       1 'SEQUENTIAL' # file access: \Z{SEQUENTIAL} | \Z{APPEND}
20 STRING FILE_STATUS       1 'UNKNOWN' # file status: \Z{UNKNOWN} | \Z{OLD} | \Z{NEW}
31 STRING BUILD_METHOD      1 'INTERNAL_COORDINATES' # method for building coordinates: \Z{INTERNAL_COORDINATE
32 STRING DIHEDRALS         0 'PHI' 'PSI' 'CHI1' 'CHI2' 'CHI3' 'CHI4' # dihedral angle type selection: \Z{phi}
33 STRING RES_TYPES         1 'ALL' # residue type selection
34 STRING ATOM_TYPES        1 'ALL' # atom type selection
35 STRING VARIABILITY_FILE  1 'undefined' # output filename
36 STRING ALIGN_CODES       0 'all' # codes of proteins in the alignment
37 STRING ATOM_FILES        0 '' # complete or partial atom filenames
38 STRING OUTPUT            1 'LONG' # what and/or how to output
39 STRING CHANGE            1 'RANDOMIZE' # what to do: \Z{RANDOMIZE} | \Z{OPTIMIZE}
40 STRING FIT_ATOMS         1 'CA' # atom type(s) being superposed
41 STRING MODEL_FORMAT      0 'PDB' # selects input atom file format: \Z{PDB} | \Z{CHARMM} | \Z{UHBD}
42 STRING SEQUENCE          1 'undefined' # protein code in the alignment whose topology is constructed
43 STRING RESTRAINT_TYPE    0 'STEREO' # restraint type to be calculated: \Z{STEREO} | \Z{BOND} | \Z{ANGLE} |
44 STRING ALIGNMENT_FORMAT  0 'PIR' # format of the alignment file: \Z{PIR} | \Z{PAP} | \Z{QUANTA} | \Z{INSIGH
45 STRING undefined81       0 ''
46 STRING ALIGNMENT_FEATURES 0 'INDICES CONSERVATION' # what alignment features to write out: \Z{ACCURACY} | \Z
47 STRING RESIDUE_TYPE      1 'undefined' #
48 STRING MATRIX_FILE       1 'family.mat' # the filename of the pairwise distance matrix
49 STRING BASIS_PDF_WEIGHT  1 'LOCAL' # a method for calculation of basis pdf weights: \Z{LOCAL} | \Z{GLOBAL}
50 STRING DISTANCE_ATOMS    2 'CA' 'CA' # atom types for distance generation
51 STRING REFERENCE_ATOM    1 '' # reference atom name in SUPERPOSE
52 STRING undefined91       0 ''
53 STRING ATOM_IDS          0 '' # atom ids: \Z{atom:residue_id[:chain_id]}
54 STRING SPHERE_CENTER     2 'undefined' 'undefined' # '\#RES1:C' 'ATOM_NAME'
55 STRING SELECTION_MODE    1 'ATOM' # selecting what: \Z{ATOM} | \Z{RESIDUE}
56 STRING SELECTION_SEARCH  1 'SEGMENT' # search method: \Z{SPHERE} | \Z{SEGMENT}
57 STRING SELECTION_STATUS  1 'INITIALIZE' # what to do with selected atoms: \Z{ADD} | \Z{REMOVE} | \Z{INITIAL
58 STRING SELECTION_SEGMENT 2 '@:@' 'X:X' # \Z{RES:CHN} ids for the first and last residues in a chain/segment
59 STRING SELECTION_FROM    1 'ALL' # selecting from: \Z{ALL} | \Z{SELECTED}
60 STRING ALIGN_CODES2     0 'all' # align codes for alignment2
61 STRING MD_RETURN         1 'FINAL' # return MODEL with \Z{MINIMAL} energy or \Z{FINAL} MODEL
62 STRING ATOM_CLASSES_FILE 1 '$(LIB)/atmcls-melo.lib' # library with atom class definitions for MODELLER non-
63 STRING RR_FILE           1 '$(LIB)/as1.sim.mat' # input residue-residue scoring file
64 STRING SEARCH_CHAINS_FILE 1 '$(LIB)/CHAINS_all.seq' # file with a list of sequence codes
65 STRING MODEL_SEGMENT     2 '@:@' 'X:X' # segment to be read in
66 STRING MODEL2_SEGMENT    2 '@:@' 'X:X' # segment to be read in
67 STRING ATOM_FILES_DIRECTORY 1 './' # input atom files directory list (e.g., \Z{dir1:dir2:dir3:./:/})
68 STRING SEARCH_SORT       1 'LONGER' # which sequence to use for normalization when sorting the hit list: \Z
69 STRING RESTRAINTS_FORMAT 1 'MODELLER' # format of the restraints file: \Z{MODELLER} | \Z{USER}
70 STRING SEARCH_CHAINS_LIST 1 '$(LIB)/CHAINS_3.0_40_XN.cod' # file with sequences
71 STRING SEGMENT_IDS       0 '' # new segment ids
72 STRING RESIDUE_IDS       0 '' # residue id (number:chnid)
73 STRING ALIGN_WHAT        1 'BLOCK' # what to align in ALIGN; \Z{BLOCK} | \Z{ALIGNMENT} | \Z{LAST}
74 STRING CLUSTER_METHOD    1 'RMSD' # what distance function to use in \C{CLUSTER}; \Z{RMSD} | \Z{MAXIMAL_DIS
75 STRING SEARCH_GROUP_LIST  1 '$(LIB)/CHAINS_3.0_40_XN.grp' # file with 40% groups of sequences
76 STRING RESTYP_LIB_FILE   1 '$(LIB)/restyp.lib' # residue type library
77 STRING SWAP_ATOMS_IN_RES 1 '' # minimize RMS by swapping atoms in these residues (1 char code: 'DEFHLNQRVY'
78 STRING ATOM_FILES2       0 '' # complete or partial atom filenames for ALIGNMENT2
79 STRING INPUT_WEIGHTS_FILE 1 '' #
80 STRING OUTPUT_WEIGHTS_FILE 1 '' #
31 LOGICAL FIT              1 on # whether to do pairwise least-squares fitting or ALIGN2D alignment

```

```

32 LOGICAL SUPERPOSE_REFINE      1 off # whether to refine the superposition
35 LOGICAL DYNAMIC_SPHERE       1 on # whether to use dynamic soft-sphere repulsion terms
36 LOGICAL DYNAMIC_LENARD       1 off # whether to use dynamic Lennard-Jones energy terms
37 LOGICAL DYNAMIC_COULOMB      1 off # whether to use dynamic Coulomb energy terms
38 LOGICAL WRITE_FIT            1 off # whether to write out fitted coordinates to .fit files
39 LOGICAL ASGL_OUTPUT          1 off # whether to write output for ASGL
40 LOGICAL ADD_RESTRAINTS       1 off # whether to add new restraints to existing restraints
41 LOGICAL ADD_SEGMENT          1 off # whether to add the new segments to the list of segments
42 LOGICAL REMOVE_GAPS          1 on # whether to remove all-gap positions in input alignment
44 LOGICAL LOCAL_ALIGNMENT      1 off # whether to do local as opposed to global alignment
45 LOGICAL WATER_IO             1 off # whether to read water coordinates
46 LOGICAL HETATM_IO           1 off # whether to read HETATM coordinates
47 LOGICAL HYDROGEN_IO          1 off # whether to read hydrogen coordinates
48 LOGICAL INITIALIZE_XYZ       1 on # whether to use IC entries to calculate all coordinates
49 LOGICAL ADD_SEQUENCE         1 off # whether to add the new sequences to the existing alignment
50 LOGICAL ALIGN3D_TRF          1 off # whether to transform the distances before dynamic programming
51 LOGICAL PATCH_DEFAULT        1 on # whether to do default NTER and CTER patching
52 LOGICAL INTERSEGMENT         1 on # whether to restrain inter-segment non-bonded pairs
53 LOGICAL ALIGN3D_REPEAT       1 off # do several starts to maximize number of equivalent positions
54 LOGICAL ALIGN_ALIGNMENT      1 off # writing out an alignment of alignments (for *)
55 LOGICAL INIT_VELOCITIES      1 on # whether to initialize velocities before MD
56 LOGICAL ADD_SYMMETRY         2 off on # whether to add segment pair, add atoms to segment pair
57 LOGICAL SPLINE_ON_SITE       1 off # whether to convert restraints to splines
58 LOGICAL ADD_PARAMETERS       1 off # whether to add new parameters to existing ones
59 LOGICAL ADD_TOPOLOGY         1 off # whether to add new residue topologies to existing ones
60 LOGICAL WRITE_WHOLE_PDB      1 on # whether to write out all lines in the input PDB file
61 LOGICAL WRITE_ALL_ATOMS      1 on # whether to write all atoms, even if unselected
62 LOGICAL CURRENT_DIRECTORY    1 on # whether to write output .fit files to current directory
63 LOGICAL DETAILED_DEBUGGING   1 off # whether to evaluate energy and derivatives wrt each restraint
64 LOGICAL DYNAMIC_PAIRS       1 off # whether to do dynamic pairs irrespective of anything
65 LOGICAL DYNAMIC_MODELLETER   1 off # whether to use dynamic MODELLER non-bonded restraints
66 LOGICAL FAST_SEARCH          1 off # whether to use fast sequence search or not
67 LOGICAL DATA_FILE           1 off # whether results go to a separate file or not
68 LOGICAL NORMALIZE_PROFILE    1 off # whether to normalize energy/violations profiles or not, by the number of t
69 LOGICAL CUT_OVERHANGS        1 off # whether to cut overhangs at OVERHANG residues or not
70 LOGICAL RESIDUE_SPAN_SIGN    1 on # whether to do N*(N-1)/2 loop for atom pairs in MAKE_RESTRAINTS RESTRAINT_T
71 LOGICAL COVALENT_CYS         1 off # whether to consider SG-SG covalent bond similar to polypeptide chain when
72 LOGICAL READ_WEIGHTS         1 off # whether to read the whole NxM weight matrix for ALIGN*
73 LOGICAL DYNAMIC_ACCESS       1 off # whether to use dynamic accessibility energy terms
74 LOGICAL DIH_LIB_ONLY         1 off # whether to use only library, not homologs for dihedral angle rsrs
75 LOGICAL NO_TER               1 off # whether to not write TER into PDB
76 LOGICAL WRITE_WEIGHTS        1 off # whether to write the whole NxM weight matrix for ALIGN*
77 LOGICAL EXCL_LOCAL           4 on on on on # whether to exclude bonds, angles, dihedrals, explicit excl pairs f
--- END OF FILE

```

The third column contains a number of values for each of the options if this number is fixed, otherwise it contains 0.

You can change any command or variable name without changing the source code relying on this file, but you can not change the order of the lines.

Index

- ACCESSIBILITY_TYPE, 60, 93
- ADD_PARAMETERS, 39, 40
- ADD_RESTRAINT**, 21, 29, 94, 95, 100, 101
- ADD_RESTRAINTS, 93, 98, 102
- ADD_SEGMENT, 40, 41
- ADD_SEQUENCE, 66, 69
- ADD_SYMMETRY, 97
- ADD_TOPOLOGY, 39
- ALIGN**, 11, 70, 73, 75, 77, 82
- ALIGN2D**, 11, 31, 35, 70, 74, 75
- ALIGN3D**, 74, 78, 81, 84
- ALIGN3D_REPEAT, 81, 82
- ALIGN3D_TRF, 81, 82
- ALIGN_CONSENSUS**, 70
- ALIGN_ALIGNMENT, 31, 70
- ALIGN_BLOCK, 30, 70, 73–77, 87
- ALIGN_CODES, 30, 40, 42, 51, 61, 64, 66, 67, 69–72, 78, 80, 81, 83, 85
- ALIGN_CODES2, 30, 67
- ALIGN_CONSENSUS**, 77
- ALIGN_WHAT, 73, 74
- alignment, 11, 14, 64
- ALIGNMENT_FEATURES, 70, 85
- ALIGNMENT_FORMAT, 28, 66, 67, 70, 85
- ARGUMENTS, 126, 129
- ASGL_2D_OUTPUT, 30
- ASGL_2D_TYPE, 30
- ASGL_OUTPUT, 80, 81, 106
- ATOM_FILES, 71
- ATOM_FILES_DIRECTORY, 35, 70
- ATOM_ACCESSIBILITY, 62
- ATOM_CLASSES_FILE, 40
- ATOM_FILES, 40–42, 50, 61, 64, 67, 69–71, 80, 81, 83
- ATOM_FILES_DIRECTORY, 35, 40–42, 47, 50, 70, 71, 80, 83, 93
- ATOM_IDS, 95, 100, 101
- ATOM_TYPES, 53, 54

- BASIS_PDF_WEIGHT, 95, 96
- BASIS_PDF_WEIGHT, 93
- BASIS_RELATIVE_WEIGHT, 93, 96
- bibliography, 2
- BIN_LIB_FILE, 93, 96
- BLOCK residues, 38
- breakpoint, 34
- bug reports, 8
- BUILD**, 28

- BUILD_METHOD**, 28, 30, 31, 48, 49
- BUILD_MODEL**, 31, 38, 41, 48, 51, 116

- CALL**, 128
- CAP_ATOM_SHIFT, 108, 110, 136
- CHANGE, 58
- CHECK_ALIGNMENT**, 116
- CHECK_ALIGNMENT**, 11, 14, 15, 27, 67
- CLOSE**, 127
- CLUSTER**, 50
- CLUSTER_CUT, 50
- CLUSTER_METHOD, 28, 50
- COLOR_ALN_MODEL**, 67
- COMMAND, 127, 129
- COMPARE**, 31, 35, 72, 73, 80, 81
- COMPARE_ALIGNMENTS**, 67–69, 74
- COMPARE_MODE, 80, 81
- COMPARE_SEQUENCES**, 12, 31
- CONDENSE_RESTRAINTS**, 21, 96, 98, 99, 101
- CONSENSUS_ALIGNMENT**, 70
- CONTACT_SHELL, 95, 106, 108, 109
- COULOMB_SWITCH, 106, 108–110
- COVALENT_CYS, 106, 108, 110
- CURRENT_DIRECTORY, 83, 84
- CUT_OVERHANGS, 70

- DATA_FILE, 85, 86
- DEBUG_FUNCTION**, 112
- DEBUG_FUNCTION_CUTOFF, 112
- DEFINE_INTEGER**, 125
- DEFINE_LOGICAL**, 125
- DEFINE_REAL**, 125
- DEFINE_STRING**, 125
- DEFINE_SYMMETRY**, 97
- DELETE_ALIGNMENT**, 87
- DELETE_RESTRAINT**, 21, 101
- DENDROGRAM**, 12, 50, 71, 72, 81, 86
- DESCRIBE**, 71
- DETAILED_DEBUGGING, 112
- DEVIATION, 56, 58, 116
- DIH_LIB_ONLY, 30, 93, 96
- DIHEDRALS, 58
- DIRECTORY, 34, 35, 39, 40, 66, 67, 72, 73, 75, 76, 85, 102, 105, 111, 125, 129
- DISTANCE_ATOMS, 42, 80, 81
- DISTANCE_RSR_MODEL, 93, 95
- DO**, 125, 128

- DO_LOOPS, 120
 DYNAMIC_ACCESS, 30, 106, 108, 109
 DYNAMIC_COULOMB, 95, 106, 108, 109
 DYNAMIC_FLAG, 29
 DYNAMIC_LENNARD, 94, 106, 108, 109
 DYNAMIC_MODELLE, 106, 108–110
 DYNAMIC_SPHERE, 94, 106, 108, 109

 ELSE, 128, 129
END_SUBROUTINE, 125, 128
END_DO, 125, 128
END_SUBROUTINE, 128
ENERGY, 12, 20, 26, 27, 30, 41, 55, 56, 89, 97, 102, 106, 110, 112, 117, 135
 ENERGY, 30
ENERGY_PROFILE, 28, 30
 EQUILIBRATE, 108, 136
 EQUILIBRATION, 110
 ERROR_STATUS, 30, 129
 evaluation, model, 12
 EXCL_LOCAL, 93, 96, 106, 108, 109
EXPAND_ALIGNMENT, 31, 84, 117
 EXPAND_CONTROL, 31, 84
 EXTEND_HOT_SPOT, 56
 EXTEND_HOT_SPOT, 55

 FAST_SEARCH, 85, 86
 FAST_SEARCH_CUTOFF, 85, 86
 FILE, 30, 35, 39, 40, 45, 47, 48, 60, 61, 66, 67, 70, 73, 85, 87, 88, 102, 103, 105, 111, 127, 129
 file naming, 34
 file types, 36
 FILE_ACCESS, 126, 127, 129
 FILE_EXISTS, 127, 129
 FILE_EXT, 34, 84, 87, 88, 126, 129
 FILE_FORMAT, 102, 103
 FILE_ID, 34, 84, 87, 88, 126, 129
 FILE_STATUS, 126, 129
 FINAL_MALIGN3D, 29, 117
 FIT, 70, 75, 78, 80–84
 FIT_ATOMS, 80–83

 GAP_EXTENSION, 31, 53
 GAP_PENALTIES_1D, 73–77, 85
 GAP_PENALTIES_2D, 75
 GAP_PENALTIES_3D, 81–84
 GAP_POSITIONS, 53
GENERATE_TOPOLOGY, 93, 94
GENERATE_TOPOLOGY, 26, 40–42, 47, 58
 GENERATE_TOPOLOGY, 116
GO_TO, 125, 127–129

 HETATM_IO, 40–42, 47, 50, 70
 HYDROGEN_IO, 40–42, 47, 50, 70

 ID1, 34, 84, 126, 129
 ID2, 34, 84, 126, 129

ID_TABLE, 12, 29, 71–73
 IDEAL_ENERGY1, 30
IF, 127, 128
INCLUDE, 35, 128
 INCLUDE_FILE, 35, 128, 129
INHERIT_DISULFIDES, 30
 INIT_VELOCITIES, 108, 110
 INITIALIZE_XYZ, 48
 INPUT_WEIGHTS_FILE, 73, 75, 77
INQUIRE, 127
 installation, 6
 INTERSEGMENT, 93, 94
 IO_UNIT, 126, 127, 129
IUPAC_MODEL, 57

 KEEP_RESTRAINTS, 29

LABEL, 125, 127
 LENNARD_JONES_SWITCH, 106, 108, 110
 LENNARD_JONES_SWITCH[2], 109
 LIBRARY_SCHEDULE, 104
 LOCAL_ALIGNMENT, 73
 LOCAL_ALIGNMENT, 73, 75, 76, 81, 83, 85

MAKE_SCHEDULE, 116
MAKE_TOPOLOGY_MODEL, 45
MAKE_REGION, 62
MAKE_RESTRAINTS, 21, 25, 26, 29, 30, 35, 38, 42, 53, 93, 99, 102, 103, 110, 116
 MAKE_RESTRAINTS, 89
MAKE_SCHEDULE, 104, 105
MAKE_TOPOLOGY_MODEL, 39, 45, 46
MALIGN, 11, 30, 74, 76, 83, 86
MALIGN3D, 11, 29, 30, 35, 50, 74, 81, 83, 117
 MALIGN3D, 11
 MATRIX_FILE, 71
 MATRIX_FILE, 71–73, 80, 81
 MATRIX_OFFSET, 73–76, 81, 83, 85
 MAX_GAPS_MATCH, 72
 MAX_GAPS_MATCH, 72
 MAX_ITERATIONS, 108, 110
 MAX_LOOP_LENGTH, 53
 MAXIMAL_DISTANCE, 93, 95
 MD_RETURN, 108, 110
 MD_TIME_STEP, 108, 110
 MDT_LIB_FILE, 93, 96
 method, 9
 MIN_ATOM_SHIFT, 110
 MIN_ATOM_SHIFTS, 108
 MIN_LOOP_LENGTH, 87
 MNCH?_LIB, 30
 MNCH_LIB, 93
 MODEL2_SEGMENT, 47
 MODEL_FORMAT, 47, 48
 MODEL_SEGMENT, 47, 61, 65
 MODEL_TOPOLOGY, 24

- MODELLER_STATUS, 30, 34, 66, 67, 73–75, 77, 81, 83, 85, 108, 110, 111
MOLPDF, 106, 108, 110
MUTATE_MODEL, 44, 53
- N_SCHEDULE, 105
NLOGN_USE, 106, 108, 110
NO_TER, 31, 48
NONBONDED_SEL_ATOMS, 29, 31, 93–96, 99, 108, 110
NORMALIZE, 30
NORMALIZE_PROFILE, 106, 107
NUMBER_LINES, 126
NUMBER_PLACES, 125–127, 129
- OBJECTS, 28, 126, 129
OBJECTS_FILE, 126, 129
OFF_DIAGONAL, 73, 75, 76, 81, 83, 85
OPEN, 126
OPERATE, 28, 126
OPERATION, 126, 128, 129
OPTIMIZATION_METHOD, 108, 109
OPTIMIZE, 29, 41, 56, 94–96, 99, 107, 108, 110, 112, 117, 135
ORIENT_MODEL, 59
OUTPUT, 28, 30, 60, 80–86, 97, 106–109
OUTPUT_CONTROL, 34, 129
OUTPUT_DIRECTORY, 35, 45, 48, 61, 69–72, 87, 88, 103, 105, 126, 127, 129
OUTPUT_WEIGHTS_FILE, 73, 75, 77
OVERHANG, 70, 73, 75, 76, 81, 83, 85
- PATCH, 20, 26, 39–41, 87
PATCH_DEFAULT, 40, 41
PATCH_DISULFIDES, 30
PATCH_SS_MODEL, 20, 30, 43
PATCH_SS_TEMPLATES, 19, 20, 30, 42, 44, 116
PICK_ATOMS, 21, 31, 38, 44, 53, 54, 78, 95, 110
PICK_ATOMS_SET, 53
PICK_HOT_ATOMS, 41, 55, 90, 110, 135
PICK_HOT_CUTOFF, 55, 56
PICK_RESTRAINTS, 21, 29, 31, 53, 56, 97, 98, 104, 105, 117
primer, modeling, 11
PRINCIPAL_COMPONENTS, 71
PRINCIPAL_COMPONENTS, 12, 73, 81
PROBE_RADIUS, 60
PROFILE_2D_PHYS, 30
program distribution, 5
program updates, 28
PSA_INTEGRATION_STEP, 60
- RADII_FACTOR, 60, 93, 106, 108–110
RAND_SEED, 48, 56, 58, 62, 85, 108, 110
RANDOMIZE, 58
RANDOMIZE_XYZ, 53, 56, 104, 116
READ, 127
- READ_ALIGNMENT, 116
READ_ALIGNMENT2, 68
READ_ALIGNMENT, 30, 66–68, 70, 71, 81
READ_ALIGNMENT2, 30, 67, 68
READ_ATOM_CLASS, 40
READ_MODEL, 38, 40, 41, 44, 47, 48, 58, 100, 116
READ_MODEL2, 47
READ_PARAMETERS, 30, 39
READ_RESTRAINTS, 21, 102, 103
READ_RESTYP_LIB, 25, 29, 39
READ_SCHEDULE, 104, 105
READ_TOPOLOGY, 39, 58
READ_WEIGHTS, 73, 75, 77
RECORD, 127, 129
REFERENCE_ATOM, 78
REFERENCE_DISTANCE, 78
REGION_SIZE, 62
REINDEX_RESTRAINTS, 101
RELATIVE_DIELECTRIC, 106, 108
REMOVE_GAPS, 66, 67, 85
RENAME_SEGMENTS, 52
RENUMBER_RESIDUES, 31, 52
RENUMBER_SEGMENTS, 31
REORDER_ATOMS, 48, 57
RES_TYPES, 53, 54
RESET, 126
RESIDUE_SPAN_RANGE, 95
RESIDUE_TYPE, 44
RESIDUE_GROUPING, 93
RESIDUE_IDS, 26, 41, 93, 95
RESIDUE_RANGE, 98, 99
RESIDUE_SPAN_RANGE, 29, 93–95, 106, 108, 110
RESIDUE_SPAN_SIGN, 29, 93–95, 110
RESIDUE_TYPE, 41, 44
RESTRAINT_PARAMETERS, 93, 95, 100
RESTRAINT_SEL_ATOMS, 29, 93, 94, 98, 99
RESTRAINT_STDEV, 93, 95
RESTRAINT_TYPE, 38, 93, 99
RESTRAINTS_FILTER, 98, 99
RESTYP_LIB_FILE, 29, 39
RESULT, 28, 126, 129
RETURN, 128
RMS_CUTOFFS, 78
RMS_CUTOFFS, 78, 80, 81
ROOT_NAME, 126
ROOT_NAME, 34, 84, 87, 88, 126, 129
ROTATE_DIHEDRALS, 53, 58
ROTATE_MODEL, 59
ROTATION_ANGLE, 59
ROTATION_AXIS, 59
ROTATION_MATRIX, 59
ROUTINE, 128, 129
RR_FILE, 72, 73, 75–77, 85, 87
- SCHEDULE_SCALE, 104, 105
SCHEDULE_STEP, 106, 108, 109

- script file, 14, 33
SEARCH_CHAINS_FILE, 85
SEARCH_CHAINS_FILE, 85, 86
SEARCH_CHAINS_LIST, 85
SEARCH_GROUP_LIST, 29, 85, 86
SEARCH_RANDOMIZATIONS, 85, 86
SEARCH_SORT, 85, 86
SEARCH_TOP_LIST, 85, 86
SEGMENT_CUTOFF, 87, 88
SEGMENT_GROWTH_C, 87
SEGMENT_GROWTH_N, 87
SEGMENT_IDS, 52
SEGMENT_MATCHING, 28, 31, 87
SEGMENT_REPORT, 87, 88
SEGMENT_SHIFT, 87
SEGMENT_SHIFTS, 87
SELECTION_FROM, 53, 54
SELECTION_MODE, 53–56
SELECTION_SEARCH, 53
SELECTION_SEGMENT, 31, 53, 65
SELECTION_SLAB, 53
SELECTION_STATUS, 53
SELECTION_STEP, 31
SEQUENCE, 40, 58
SEQUENCE_COMPARISON, 72, 73
SEQUENCE_COMPARISON, 29, 71, 72
SEQUENCE_SEARCH, 11, 29, 30, 85
SEQUENCE_TO_ALI, 44, 69
SET, 35, 41, 125
SET NLOGN_USE, 111
SHEET_H-BONDS, 95
SIGNIF_CUTOFF, 29, 30, 85, 86
SMOOTHING_WINDOW, 106, 107
SPHERE_CENTER, 53
SPHERE_RADIUS, 53
SPHERE_STDV, 106, 108
SPLINE_MIN_POINTS, 102
SPLINE_MIN_POINTS, 102
SPLINE_DX, 102
SPLINE_MIN_POINTS, 102
SPLINE_ON_SITE, 93
SPLINE_RANGE, 102
SPLINE_RESTRAINTS, 102
SPLINE_SELECT, 102
STOP, 125, 129
STOP_ON_ERROR, 34, 66, 67, 73, 75–77, 81, 83, 85,
 108, 110, 129
STRING_ARGUMENTS, 126, 129
STRING_IF, 129
STRING_OPERATE, 126
SUBROUTINE, 128
SUPERPOSE, 29, 48, 78, 81
SUPERPOSE_REFINE, 78
SWAP_ATOMS_IN_RES, 29, 78
SWITCH_TRACE, 109, 111
SYMMETRY_WEIGHT, 97
SYSTEM, 127
TEMPERATURE, 108, 110
 templates, 11, 12
THEN, 128, 129
TOPOLOGY_MODEL, 39–45, 60, 93, 106, 108, 109
TRACE_OUTPUT, 108, 109, 111
TRANSFER_RES_NUMB, 48
TRANSFER_RES_NUMB, 51
TRANSFER_XYZ, 28, 38, 41, 48, 50, 83, 116
TRANSLATION, 59
 tutorial, 14
UPDATE_DYNAMIC, 106, 108–110
VARIABILITY_FILE, 72
VARIABLES, 125, 129
VIOL_REPORT_CUT, 55, 106, 107
VIOL_REPORT_CUT2, 106, 107
WATER_IO, 40–42, 47, 50, 70
WRITE, 28, 126
WRITE_ALIGNMENT, 28, 44, 70, 74, 75, 77, 83,
 86
WRITE_ALL_ATOMS, 48
WRITE_DATA, 30, 60, 62
WRITE_FIT, 11, 83, 84
WRITE_MODEL, 48, 116, 117
WRITE_MODEL2, 48
WRITE_PDB_XREF, 61
WRITE_RESTRAINTS, 103, 116
WRITE_SCHEDULE, 104, 105
WRITE_TOP, 127
WRITE_TOPOLOGY_MODEL, 45
WRITE_WEIGHTS, 73, 75, 77
WRITE_WHOLE_PDB, 83, 84