The CCP4 Suite

— Computer programs for
protein crystallography

Overview and manual

Edition of February 2006
(also available as part of the CCP4 distribution)

Copyright © 1993 – 2006 The CCP4 Project, CCLRC Daresbury Laboratory

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by CCP4.

You can get extra printed copies of this document from Daresbury at cost.

The Secretary to CCP4
CCLRC Daresbury Laboratory
Warrington WA4 4AD
UK

Telephone: (+44) 1925 603929 (direct line)
Facsimile: (+44) 1925 603825
e-mail: ccp4@ccp4.ac.uk

Contents

Preface **ix**

I Introduction 1

1 Introduction **1**
1.1 CCP4 constitution . . . 1
1.2 Philosophy of the CCP4 suite . . . 1
1.3 Referencing CCP4, etc. . . . 2

II Tutorial material 3

2 Overview **5**
2.1 Steps in structure solution . . . 5
2.2 Program overview . . . 7
2.2.1 Data processing . . . 7
2.2.2 Data scaling and reduction . . . 7
2.2.3 Data combination and scaling different sets . . . 8
2.2.4 Obtaining ab initio phases . . . 8
2.2.5 Map and structure factor calculation . . . 10
2.2.6 Molecular averaging and phase improvement . . . 10
2.2.7 Map interpretation and manipulation . . . 10
2.2.8 Refinement of protein models . . . 11
2.2.9 Coordinate analysis . . . 11
2.2.10 Pictorial presentation of results . . . 12
2.2.11 Utility programs . . . 13
2.2.12 Data Harvesting Utilities . . . 13
2.2.13 Graphical user interface . . . 14
2.2.14 Other systems . . . 14

3 General CCP4 usage **15**
3.1 Program documentation . . . 15
3.2 Running the programs . . . 15
3.2.1 Command line arguments/file connexion . . . 15
3.2.2 Keyworded input . . . 16
3.3 Examples . . . 18
3.3.1 Unix . . . 18
3.4 Output . . . 19
3.5 Rudiments of reflexion files . . . 19
3.6 Transporting data files . . . 20
3.7 Details of logical name processing etc. . . . 20
3.7.1 Command line . . . 22
3.8 Graphical User Interface . . . 22

4 Data processing and reduction **23**
4.1 Data processing . . . 23
4.2 Data reduction . . . 23

5 Manipulating and Displaying Data Files **25**
5.1 *hkl* manipulation . . . 25
5.2 Map file utilities and plotting . . . 25
5.3 Interpretation and manipulation of coordinates . . . 26
5.3.1 Converting between fractional and orthogonal coordinates . . . 26
5.4 Plotting . . . 27

6 Isomorphous Replacement **29**
6.1 MIR structure solutions: steps and programs . . . 30
6.2 Combination of phase information . . . 31
6.3 Background . . . 31
6.4 Determination of heavy atom positions . . . 32
6.4.1 Scaling between native and derivative data . . . 33
6.4.2 Estimation of F_H . . . 33
6.4.3 Determination of the heavy atom positions . . . 34
6.5 Refinement of heavy atom parameters . . . 36
6.5.1 "Maximum Likelihood Phase" refinement . . . 36
6.5.2 Vector space refinement . . . 37
6.5.3 "F_H" refinement . . . 37
6.6 Phase determination . . . 38
6.6.1 Isomorphous replacement data . . . 38
6.6.2 Anomalous scattering data . . . 38
6.6.3 Multi-wavelength Anomalous scattering Data . . . 38
6.6.4 Density modification . . . 39
6.6.5 Treatment of errors . . . 39
6.6.6 Hendrickson–Lattman coefficients . . . 40
6.6.7 Single Isomorphous Replacement . . . 40

7 Molecular Replacement **41**
7.1 Molecular replacement: steps and programs . . . 41
7.1.1 Detection of non-crystallographic symmetry using self rotation function . . . 42
7.1.2 Cross rotation function . . . 42
7.1.3 Translation function . . . 42
7.1.4 Checking the results . . . 42
7.2 Introduction . . . 42
7.3 The rotation function . . . 43
7.3.1 Model . . . 43
7.3.2 Resolution limits . . . 44
7.3.3 Patterson integration radius . . . 44
7.3.4 Model unit cell size . . . 45
7.3.5 Omitting weak terms . . . 45
7.3.6 Normalising . . . 45
7.3.7 Origin Removal . . . 45
7.3.8 Non-crystallographic symmetry . . . 45
7.4 The translation search . . . 46

7.4.1 *R* factor search . . . 47
7.4.2 Translation functions . . . 47

8 Phase Improvement **51**
8.1 Density Modification . . . 51
8.1.1 The Problem . . . 51
8.1.2 The Method . . . 51
8.2 Using `dm` . . . 52
8.3 Using Solomon . . . 54
8.4 Estimating solvent content . . . 54

9 Refinement and validation **57**
9.1 Least squares structure refinement . . . 57
9.1.1 Comparison of `restrain` and `prolsq` . . . 57
9.2 Maximum Likelihood refinement . . . 58
9.2.1 TLS refinement in `refmac5` . . . 58
9.3 Automated model building . . . 58
9.4 Difference map generation . . . 59
9.5 Why is protein refinement difficult? . . . 59
9.6 Free *R* factor . . . 60
9.7 Validation, gross and overall errors . . . 60
9.7.1 Validation . . . 60
9.7.2 Errors . . . 60
9.7.3 Bad practice . . . 61

10 The Fast Fourier Transformation **63**
10.1 Introduction . . . 63
10.2 How it works . . . 63
10.2.1 Example . . . 64
10.2.2 Why is it fast? . . . 64
10.3 Crystallographic FFT . . . 65
10.4 Programs . . . 65

III The propagation, care and feeding of a CCP4 installation **67**

11 Installation **69**
11.1 First off . . . 69
11.2 Directory structure . . . 69
11.3 Building under Unix . . . 70
11.3.1 Preliminaries . . . 70
11.3.2 Unpacking the files . . . 70
11.3.3 Environment variables and `ccp4.setup` . . . 70
11.3.4 Configuration . . . 71
11.3.5 Building . . . 71
11.3.6 Testing . . . 72
11.3.7 Problems . . . 72
11.3.8 Saving space and shared libraries . . . 72
11.3.9 Installing updates . . . 73
11.3.10 Installation summary . . . 73

11.4 CCP4I . . . 73
11.5 X-windows programs . . . 74
11.6 The full story on `configure` . . . 74

12 Support, bug reports etc. 77
12.1 Release policy . . . 77
12.2 Support and bug reports . . . 77
12.2.1 Reporting bugs . . . 78

13 Resources 79
13.1 CCP4 Web Pages . . . 79
13.2 Anonymous ftp . . . 79
13.3 Electronic mailing lists . . . 80
13.3.1 The CCP4 Bulletin Board . . . 80
13.3.2 The Bulletin Board Subscriber Service . . . 80
13.3.3 The developers' list . . . 80
13.3.4 Summary . . . 80
13.4 Other crystallographic software . . . 80
13.5 Crystallographic information/discussion . . . 81
13.6 Sources of general free software . . . 81
13.6.1 Free compilers . . . 82

IV Hackers' bit 83

14 Writing and contributing programs 85
14.1 General advice . . . 85
14.2 Requirements for CCP4 code . . . 85
14.3 Non-portable features . . . 86
14.4 Stealing code . . . 87
14.5 Using the pre-processor in programs . . . 87
14.6 Library modules . . . 88

15 Porting CCP4 89
15.1 Portability features . . . 89
15.2 Fundamental operating system dependencies . . . 89
15.3 Unix-flavour dependencies . . . 89

16 File formats 91
16.1 Reflexions (MTZ) . . . 91
16.1.1 Orientation data . . . 92
16.1.2 Standard column names and types . . . 93
16.1.3 Missing Data Treatment . . . 93
16.2 Maps . . . 95
16.3 Coordinates . . . 97
16.4 PLOT84 graphics files . . . 97
16.5 Library data files . . . 98
16.5.1 Symmetry operators: `syminfo.lib` . . . 98

V Appendices **99**

References **101**

Program index **107**

Index **113**

Preface

> *Alice in Wonderland: the best book on computer science for the layman.*
> *The best book on* anything *for the layman.*
> *— Anon., Usenet book recommendations.*[1]

This manual attempts to provide documentation of the CCP4 program package at a level above that of detailed input requirements for individual programs. Sadly over the years material has become incomplete in places. Suggestions for improvement are welcome, particularly in the form of new passages for inclusion!

Although there is tutorial material included here, novices will need background material e.g., from textbooks like (Blundell & Johnson, 1976; Glusker & Trueblood, 1985; Luger, 1980; Stout & Jensen, 1989; Drenth, 1994; Giacovazzo *et al.*, 1992; Perutz, 1992; Rhodes, 1993) and sources such as (Hahn, 1992).

It is also worth noting that the HTML formatted documentation included with the CCP4 suite itself contains much supplementary material - see for example the documents under the `General` section of the main program index.

Cross-references in the text of the form '§*n.m*' refer to chapter *n*, section *m* etc.

Credits

This document contains material from various sources. It is mostly based on the previous manual (which was mainly an installation guide) and Wojtek Rypniewski's 'overview' `man` 'page'. The previous manual was updated by Dave Love from a version containing contributions from Peter Daly, Kim Henrick and Wojtek Wolf. Chapter 1 contains material originating from Phil Evans and John Campbell. Chapter 6 was originally based on material by Silvia Onesti. Some of §8 was taken from material by Andrew Leslie and Kevin Cowtan. The description of the MTZ file format was taken from Sandra Mc Laughlin's `mtzlib` documentation and that for `maplib` from John Campbell and Phil Evans'. Several of the sections outlining procedures and program usage are taken from lecture notes by Eleanor Dodson and often have accompanying example files (in the `examples` directory) derived from that course.

Wojtek Rypniewski's acknowledgements for what he wrote include: "I had asked various people for help and comments. I would like to thank Ian Tickle for reading very thoroughly the chapter on molecular replacement and making several corrections. Fred Antson did the same for molecular averaging. Sandra McLaughlin helped me write some guidelines on programming standards."

Dave Love edited, designed and typeset this document. Subsequent updates from Martyn Winn and Peter Briggs.

Many people have contributed both to the software and documentation—see the program writeups—but Eleanor Dodson Phil Evans and Ian Tickle deserve special mention.

[1] The quotations by or relating to Lewis Carroll have some significance because Carroll (Dodgson) was born at Daresbury, administrative base of CCP4. Also the original editor was never previously far from either Looking Glass House or Dodgson's college.

Part I

Introduction

Chapter 1. Introduction

"Where do you come from?" said the Red Queen,
"And where are you going?"
— Through the Looking Glass

1.1 CCP4 constitution

The CCP4 program suite is a collection of disparate programs covering most of the computations required for macromolecular crystallography. They have been collected and developed under the auspices of the Collaborative Computing Project Number 4, in Protein Crystallography, supported by the UK Science and Engineering Research Council (SERC) since 1979 and currently the Biotechnology and Biological Sciences Research Council (BBSRC), and coordinated at Daresbury Laboratory. The Project aimed to support collaboration between those working on such software in the UK, and to assemble a comprehensive collection of it to satisfy the computational requirements of the relevant UK groups. The results of this effort are also made available for distribution to academic and commercial users outside the UK. The distribution, described herein, is often loosely referred to as 'CCP4', but is properly 'The CCP4 Suite'.

CCP4 is overseen by two committees: Working Group 1, largely comprising heads of groups, normally meets annually and makes general policy. Working Group 2 meets more frequently and coordinates the developers. The current chairmen of WG1 and WG2 are Jim Naismith (University of St. Andrews) and Phil Evans (MRC LMB) respectively. Working Group 2 has a representative of industrial users, currently Tadeusz Skarzynski of GlaxoWellcome. The various CCPs are coordinated by a steering committee. The core activities of the CCP4 project are run by a group at Daresbury Laboratory, which currently (February 2006) comprises Martyn Winn, Peter Briggs, Charles Ballard, Francois Remacle, Norman Stein, Daniel Rolfe, Ronan Keegan and Maeri Howard. CCP4 also fund a number of developers in the UK through the BBSRC grant and receipts from industry.

1.2 Philosophy of the CCP4 suite

Unlike many other packages, particularly for small molecule crystallography, the CCP4 suite is a set of separate programs which communicate via standard data files, rather than all operations being integrated into one huge program. This has some disadvantages in that it is less easy for programs to make decisions about what operation to do next—though it is seldom a problem in practice—and that the programs are less consistent with each other (although much work has recently been done to improve this). The advantage of loose organisation is that it is very easy to add new programs or to modify existing ones without upsetting other parts of the suite. This is the approach successfully taken by Unix. Converting a program to use the standard CCP4 file formats is generally straightforward, and the philosophy of the collection has been to be inclusive, so that several programs may be available to do the same task. The components of the whole system are then a collection of programs using a standard subroutine library to access standard format files. Most of the programs are written in standard FORTRAN77.

To use the programs the user must assign input and output files, including library and scratch files where necessary (though defaults are usually defined), and run the programs. Often an output file becomes the input to the next step, and system parameter substitution may be used to create filenames in a systematic way. Most crystallographic calculations involve a series of steps in which no decisions need be made until the end, and a command file provides an easy way of chaining calculations. A graphical user interface is now also provided as a way of facilitating running of the programs.

Standard file formats are defined for the principal sorts of data used in crystallography: reflection data; electron density maps; and atom coordinates. In defining these formats, a number of trade-offs have to be made between efficiency (in space and access time), flexibility, portability, and simplicity of use. Since the data formats form the core of the suite, they are described in more detail in § 16. The formats used in the CCP4 suite share many characteristics with those used in other packages.

The programs are distributed in source form, so they can be studied (don't look *too* closely!), modified and fixed by recipients, in contrast to most commercial software. In addition, executables are distributed for some common platforms. Otherwise, you will need Fortran, C and C++ compilers to build the suite, although there are freely-obtainable Fortran and C/C++ implementations (see §13.6). There are freely-available Unix implementations for PC-type boxes at least, and the combination of a sufficiently well-resourced top-end PC and, say, the Linux operating system and compilers reputedly makes a competent environment for scientific computing.

There is a policy of continual technical and scientific updates to the suite. Where existing programs have been incorporated into the suite they have often subsequently undergone considerable modification above that needed to use the CCP4 file formats.

1.3 Referencing CCP4, etc.

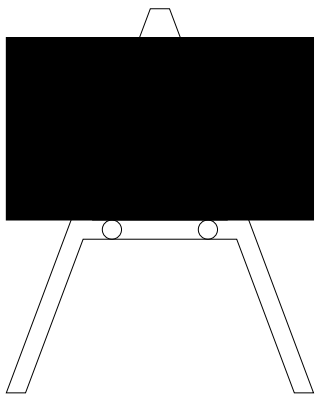
Please see the distribution conditions for the suite (available from the CCP4 website at http://www.ccp4.ac.uk/ccp4license.html). In particular, please note that any publication arising from use of the CCP4 program suite should include a reference to (Collaborative Computational Project, Number 4, 1994) given as:

> COLLABORATIVE COMPUTATIONAL PROJECT, NUMBER 4. 1994. "The CCP4 Suite: Programs for Protein Crystallography". *Acta Cryst.* **D50**, 760–763.

Such citations may be valuable to us in the future in maintaining funding. In addition, authors of specific programs should be referenced where applicable—see the program's documentation and/or output.

Part II

Tutorial material



Chapter 2. Overview of the crystallographic process and CCP4 programs

"What is it you want to buy?" the Sheep said at last, looking up for a moment from her knitting.
"I don't quite *know yet," Alice said very gently. "I should like to look all round me first, if I might."*
— Through the Looking Glass

2.1 Steps in structure solution

The basis of any crystallographic project is *well diffracting crystals* which allow the diffraction pattern to be measured accurately. This can be achieved in a variety of ways depending on the type of structure we are attempting to solve, the availability of equipment and on the personal preferences and prejudices of the investigator.

The basic choice in data collection is between diffractometers and area detectors (including photographic film and image plates). Area detectors are usually more suitable for proteins. Most machines provide their own software for data collecting. The raw data are integrated and suitable corrections are made. The relevant software is briefly summarised in §2.2.1.

Once we have extracted *integrated intensities* from the images the responsibility of CCP4 is to pick up these intensities and provide tools for further processing and analysis. The data are scaled and merged. Corrections are applied, the data are then scaled together and reduced to the asymmetric unit. Finally the intensities are converted to *structure factor amplitudes*. The relevant software is summarised in §2.2.1.

From this point solving the structure is nothing other than finding the *phases* i.e., the other 50% of the information we need but cannot measure directly.

The fundamental technique for obtaining phases ab initio is the *multiple isomorphous replacement* (MIR). It involves introducing strong scatterers (heavy atoms) into the crystal lattice. If, as a result, the lattice was not significantly damaged, the presence of the heavy atom should cause small but measurable changes in intensities. Several different heavy atom derivatives are usually needed. Data from them are collected and finally the various data sets are scaled together using the programs summarised in §6.

The Patterson difference map is calculated for each derivative to see if the heavy atom positions can be determined for any of them. This is usually only possible if there are a few of the heavy atoms. In most cases this is done manually although programs for automatic peak interpretation or direct methods can also be used. Once the major heavy atom sites for one of the derivatives are determined they can be used to calculate preliminary phases. The phases then can be used to solve for heavy atom positions in the other derivatives. Next the positions and occupancies of heavy atoms for all the derivatives can be refined simultaneously to give an improved set of phases. It is an iterative process during which more heavy atom sites can be found (usually with low occupancy) and included in the refinement. Finally a statistically best set of phases is obtained. The relevant programs are summarised in §2.2.4.1.

If we already have a reasonable model of the structure we are trying to solve, e.g. from a homologous protein or the same protein in a different crystal lattice, we may attempt a method known as *molecular replacement*. As more and more structures are solved this is an increasingly successful method of determining preliminary phases. First the molecule's orientation in the crystal unit cell is determined from a cross rotation function. If the molecule has an internal (non-crystallographic) symmetry the orientation of the symmetry elements can be determined from a self rotation function and checked for consistency with the cross rotation. The translational parameters are then obtained from an R factor search or from a translation function. The relevant programs are summarised in §2.2.4.2.

In case of small molecules the phases are usually obtained by *direct methods*. These have been successful with structures of up to about 150 atoms. However, they have sometimes been used to find positions of the heavy atoms in some difficult cases of heavy atom derivatives. The programs are summarised in §2.2.4.3.

A basic crystallographic tool is the *Fourier transform*. Using it, Patterson maps are calculated from structure factor amplitudes and electron density maps are calculated from structure factor amplitudes and phases (transformations from reciprocal space to real space). Quite often one also has to calculate structure factors from coordinates via an 'F_{calc} map' (real space to reciprocal). Most of the relevant programs use the fast Fourier transform FFT algorithm. They are summarised in §2.2.5.

One may attempt to improve the phases by *density modification*. A variety of methods can be used to modify the electron density map which are summarised in §2.2.6. For structures where there is a non-crystallographic symmetry we can also use *molecular averaging*. This is a powerful means of improving phases, especially in cases of high non-crystallographic symmetry, but if carefully used in combination with solvent flattening even a two-fold averaging can lead to a substantial improvement of the map. One first has to determine the symmetry transformations for averaging, either from heavy atom positions or from molecular replacement. Then a molecular envelope is determined either automatically or from an existing model. Cycles of averaging and solvent flattening are carried out until convergence is reached. The relevant software is summarised in §2.2.6.

Having satisfied ourselves that we have obtained the best possible phases we should now attempt to *interpret the electron density map*. Tracing the polypeptide chain of a new structure and the subsequent readjustments are very demanding on both good judgement and time. It is therefore very important that the graphics software we use is convenient to use and reliable. It is summarised in §2.2.7,2.2.10.

Next follows a process of *refinement*. If our starting model comes from molecular replacement it is likely to contain large systematic errors in the coordinates. A preliminary step of rigid body refinement should be carried out to reduce the largest errors in coordinates. In this technique the model is divided into a number of constrained 'rigid bodies'. The number of parameters refined here is relatively small, so one can refine using low resolution data and thus increase the radius of convergence. The 'classical' restrained least-squares refinement has a relatively small radius of convergence. This is because it uses a linear approximation to a strongly non-linear problem. Discrepancies are minimised between the observed structure factors and those calculated from the model. The positions of the atoms are shifted while maintaining geometric constraints on the known bond lengths, angles and Van der Waals contacts. Many cycles of refinement are usually required as well as several rounds of map inspection and manual readjustments of the model. An alternative to least-squares refinement is to minimise a residual based on maximum likelihood ideas. In recent years techniques employing molecular dynamics in crystallographic refinement have become increasingly popular. Thermal motions are simulated to help the refinement overcome the local minima. The programs used in refinement are summarised in §2.2.8.

Throughout the process of refinement one often needs to assess the quality of the model. Various programs for the *analysis of coordinates* are summarised in §2.2.9.

The complexity of macromolecules makes clear *pictorial presentation* of their structures particularly tricky. The plotting programs are summarised in §2.2.10.

CCP4 provides a collection of useful utility programs which may be used at various stages, including those useful in collecting together information for deposition of macromolecular structures (*data harvesting*). There is also a *graphical user interface* to many of the programs.

2.2 Program overview

Programs mentioned below which are marked with an asterisk are not part of the CCP4 suite (although this classification is not always well defined) but are listed here for completeness. Some of these are what we term "aggregated" to the suite (distributed with CCP4), and some are completely separate ("external") packages not distributed with CCP4. The tutorial chapters have more information on the use of these programs. See also the program index (page 107) and § 13.4 for information on obtaining some non-CCP4 programs.

2.2.1 Data processing

2.2.1.1 Film/image plates/CCDs

mosflm `mosflm` is a widely used data processing program for image plate, CCD and oscillation data.

denzo* film and image plate processing package (Otwinowski, 1993).

d*trek* Image processing package.

LAUE* For processing data taken with the Laue method. (Aggregated.)

ipdisp For viewing and measuring images under X-Windows.

hklview displays zones of reciprocal space as pseudo-precession images under X-Windows.

rotgen Simulate X-ray diffraction rotation images.

2.2.1.2 Other area detectors

madnes* Package for processing FAST area detector data (and data from other detectors). (See also CCP4 programs `undupl` and `madlat`.)

xentronics* Proprietary program for processing Xentronics data.

XDS* Kabsch's program for processing Xentronics data.

2.2.1.3 Diffractometers

Most machines provide their own software and produce an ASCII file. `f2mtz` or `combat` will convert this into MTZ format respectively for merged and un-merged data.

2.2.2 Data scaling and reduction

combat is a *jiffy*[1] for converting various foreign formats of un-merged intensity data to multi-record MTZ format for input to `scala`. It can easily be extended to accept additional formats.

scalepack2mtz Converts merged scalepack output into MTZ format.

detwin Detwins reflection data from merohedrally twinned crystals.

dtrek2mtz Converts d*trek scalemerge output into MTZ format.

dtrek2scala Converts integrated intensity and header files from d*trek into multi-record MTZ format for input to `scala`.

[1] 'Jiffy' has been adopted in this community to mean a small utility program c.f., mainstream jargon use as a unit of time (Raymond, 1993).

postref postrefinement of film data.

scala Replaces rotavata and agrovata. Scales batches of data from procesed images together in a similar way to (Kabsch, 1988). Applies scale factors, adds together partially recorded reflexions, monitors and rejects bad agreements between repeated measurements or symmetry equivalents and averages them for output. Various statistics on the averaging are produced.

truncate Converts from intensities to Fs by the method of (French & Wilson, 1978) and does a Wilson plot to estimate B factor.

unique Generates a unique list of reflexions.

uniqueify Script for completing a dataset up to a high resolution limit, and adding free-R flags.

wilson Makes a Wilson plot.

rebatch Alters batch numbers in an unmerged MTZ file (saves running `mosflm` again).

2.2.3 Data combination and scaling different sets

cad Combines Assorted Data from several MTZ files and resorts, changes asymmetric unit etc. See also `mtzutils`.

scaleit scaling of derivative to native data with anisotropic temperature factor; see also `fhscal`.

icoefl Scaling F_o to two or more F_cs (for bulk solvent etc.).

rstats Least squares scaling between F_o and F_c.

fhscal Scales native to derivative data.

2.2.4 Obtaining ab initio phases

2.2.4.1 Heavy atoms (MIR, MAD, SAD)

(See also scaling by `fhscal` and `scaleit` above.)

abs Determines the absolute configuration (hand) of the heavy atom substructure.

vecsum Patterson search peak search.

vectors Generates all Patterson vectors from a list of input atoms and produces a list of all vectors which fall within the volume of the Patterson calculated.

havecs Generates Patterson vectors from fractional sites, writing them as orthogonal coordinates in Brookhaven format.

mlphare Calculates phases and refines heavy atom parameters; for refinement of parameters see also `vecref`. See also `SHARP`.

oasis Program for breaking phase ambiguity in OAS or SIR, from Quan Hao

professs Determination of NCS operators from heavy atoms (Kevin Cowtan).

findncs Detect NCS operations automatically from heavy atom sites.

vecref Refines heavy atom parameters in vector space. See also `mlphare`.

sapi use direct-methods program to find heavy atom sites.

sigmaa Phase combination of isomorphous and calculated phases, calculation of Sim weight, etc. (Read, 1986).

MULTAN* Direct methods package. Can be used for finding heavy atom sites from the Patterson map.

SHELX-97* Direct methods package. Can be used for finding heavy atom sites using isomorphous differences. (see http://linux.uni-ac.gwdg.de/SHELX/)

rsps Determines heavy atom positions from derivative difference Patterson maps. Can be used interactively to examine the fit of potential sites to the map.

crossec tabulates anomalous scattering factors f' and f''.

SHARP* Statistical Heavy-Atom Refinement and Phasing program. Probably the program of choice for heavy atom refinement and phasing. (see http://babinet.globalphasing.com/)

BP3 Multivariate likelihood substructure refinement and phasing of S/MIR(AS) and SAD. See also `mlphare`.

2.2.4.2 Molecular replacement

amore is a complete molecular replacement system in one program, from Jorge Navaza.

molrep automated program for molecular replacement, from Alexei Vagin. (Vaguine *et al.*, 1999)

phaser-MR is crystallographic software for phasing macromolecular crystal structures with maximum likelihood techniques. This is the molecular replacement module, which replaced `beast`.

beast brute-force molecular replacement with Ensemble Average Statistics, Maximum likelihood-based molecular repaclement (Randy Read). Obsoleted by `phaser-MR`

bulk applies bulk solvent correction for the translation search and rigid body refinement steps of AMoRe.

fsearch performs up to 6 dimensional molecular replacement solution given a predetermined envelope.

ecalc calculate normalized structure amplitudes Es from Fs.

chainsaw model preparation.

almn Crowther fast rotation function. Obsoleted by `amore`.

polarrfn Kabsch's fast polar rotation function with plot.

stnet, stgrid Generate stereographic net plots for use with polarrfn.

rfcorr analyzes correlations between cross- and self-rotation functions

getax Real space searching for rotation axis of a $D\langle n\rangle$ or $C\langle n\rangle$ multimer.

tffc space group general translation function.

mapsig peak search and statistics on signal/noise for translation function map. Also sum, product, ratio of two maps.

rsearch R factor search.

MERLOT* a complete molecular replacement package by Paula Fitzgerald (Fitzgerald, 1988).

CNS*

rotmat gives `X-PLOR`/`MERLOT`/`amore` equivalent rotation angles.

2.2.4.3 Direct methods

acorn ab initio procedure for the determination of protein structure at atomic resolution (Yao Jia-Xing).

rantan Direct Method module for the determination of heavy atom positions in a macromolecule structure or to determine a small molecule structure.

sapi

SHELX-97* Comprehensive package of ab initio structure solution, Patterson peak analysis, unconstrained refinement etc.

MULTAN* Find phases by direct methods. Most useful for finding heavy atom sites.

HySS* Hybrid Substructure Search.

2.2.4.4 MAD phasing

mlphare
abs
madsys* 'Geometric' approach by Wayne Hendrickson/B. Weis
SHARP* Statistical Heavy-Atom Refinement and Phasing program. (see http://babinet.globalphasing.com/)
revise estimates FM using MAD data, where FM is an optimised value of the normalised anomalous scattering.
BP3 Multivariate likelihood substructure refinement and phasing of S/MIR(AS) and SAD.

2.2.5 Map and structure factor calculation

sfall Structure factor calculation using inverse FFT.
fft Map calculation using fast Fourier algorithm.
mapmask Extend asymmetric unit of mask or map to cover any grid volume.
mapdump Print map header and part or all of map.

2.2.6 Molecular averaging and phase improvement

cpirate Statistical Phase Improvement from Kevin Cowtan.
dm Density modification using solvent flattening, Sayre's equation, histogram matching, NCS averaging and iterative skeletonisation.
dmmulti A multi-crystal version of DM.
solomon Modifies the electron density maps by averaging, solvent flipping and protein truncation.
demon/angel* Fred Vellieux's molecular averaging and solvent flattening package; there are versions for both Groningen BIOMOL files and CCP4 MTZ files.
RAVE* The Uppsala averaging package.

2.2.7 Map interpretation and manipulation

ccp4mg The CCP4 molecular graphics program. Optimised for the generation of presentation graphics and surfaces.
COOT The Crystallographic Object Orientated Toolkit, for model building, completion and validation.
xdlmapman Manipulates maps, bones skeletonisation, exchanges formats etc. (Kleywegt & Jones, 1996a).
bones2pdb Convert a `bones` output file to to PDB file for `ncsmask`.
extends extend Fourier maps and the compute standard uncertainty of electron density
fffear A package which searches for molecular fragments in poor quality electron density maps.
O* General model building/manipulation/display graphics program (Jones *et al.*, 1991).
OOPS* Facilitates process of rebuilding protein structure inside electron density (Kleywegt & Jones, 1996b).
mama2ccp4 Convert RAVE/MAMA-format masks to CCP4 format.
map2fs Convert a CCP4 map to XtalView fsfour format.
mapmask map/mask extend program
maprot map skewing, interpolating, rotating and averaging program
mapsig can do arithmetic on two maps.
omit calculate omit-maps according to Bhat procedure.
ncsmask Performs operations on non-crystallographic symmetry masks, e.g. before `dm`.
RAVE* and associated programs (see above).
PyMOL* Python enhanced molecular graphics program (see http:/pymol.sourceforge.net).

overlapmap Map summation (averaging) and subtraction, real-space correlation coefficients and *R* factors.

peakmax Pick peaks on map (e.g., for searching for water).

2.2.8 Refinement of protein models

refmac5 Refine or idealize structures, using intensity or amplitude based least squares or -loglikelihood residuals. (Murshudov *et al.*, 1997)

libcheck Generates and manages the library files which provide complete chemical and geometric descriptions of residues and ligands used in refinement by `refmac5`. `sketcher` is part of ccp4i and provides a graphical interface to libcheck.

arp_waters version 5.0 of the Automated Refinement Procedure. (Lamzin & Wilson, 1992). See http://www.embl-hamburg.de/ARP/ for the most recent version of `arp_warp`.

restrain, tlsanl restrained geometry, rigid body, use of amplitude and phase observations, group anisotropic displacement parameters, disordered solvent corrections (Driessen *et al.*, 1989)

rdent Create dictionary entries for `restrain` from PDB file.

CNS* X-ray and molecular dynamics refinement, also for 'rigid body'.

TNT* Ten Eyck and Tronrud's refinement package, also for 'rigid body'.

SHELX-97* Least squares refinement.

2.2.9 Coordinate analysis

act Coordinate checking.

angles Calculates angles and bond lengths, Ramachandran plot.

anisoanl Analyses of anisotropic displacement parameters.

areaimol Finds solvent accessible area (or area differences) of atoms in a Brookhaven coordinate file (or files).

baverage Average *B* values for main and side chain atoms. Very useful program which gives average r.m.s. *B*s for main and side chain atoms. Much simpler alternative to Branden real space *R* factor. Can be used to truncate *B*s to exclude wildly too small or too high values. See also `overlapmap`.

bplot Plots B-factors by residue.

cavenv Calculates cavities in macromolecular structures.

compar Compares two or three sets of atomic coordinates. R.m.s. differences as a function of residue, atom type and *B* values.

contact Calculates various types of contacts and analyses water hydrogen bonding. (See also `act`.)

distang Calculates intra- and inter-molecular distances.

dyndom Determines dynamic domains when two conformations are available. (Hayward & Berendsen, 1998)

dssp* Definition of secondary structure; produces dictionary of secondary structure (Kabsch & Sander, 1983).

gensym Generates all symmetry-related sites from a list of input atoms, and produces a list of all sites which fall within the volume of the defined volume.

geomcalc Does various geomtry calculations on a molecule.

hgen Generates hydrogen atoms for a protein coordinate file with standard geometry.

lsqkab Least squares fit of two sets of coordinates.

ncont Analyses contacts between subsets of atoms in a PDB file.

pdbcur A curation tool providing various analyses and manipulations of PDB files.

pdbset Various useful manipulations of PDB files: e.g., add CRYST and SCALE lines, generate symmetry-related subunits, rename chains, renumber residues, transform coordinates.

polypose Superposition of multi-domained structures.

procheck Comprehensive stereochemistry checking.

rotamer List amino acids whose side chain torsion angles deviate from Richardson's Penultimate Rotamer Library (Dirk Kostrewa).

rwcontents Count atoms by type, and other analyses.

sc Program to analyse shape complementarity of molecular interfaces.

sfcheck Program for assessing agreement between atomic model and X-ray data

sortwater Sorts waters by the protein chain to which they "belong" in the case of a protein with several equivalent subunits.

surface Determines accessible surface area.

topp An automatic topological and atomic comparison program for protein structures.

volume Determines polyhedral volume around selected atoms.

watertidy Assigns waters to nearest subunit and residue.

watncs Pick waters which follow NCS and sort out to NCS asymmetric unit.

watpeak Lists peaks found by peakmax near to atoms.

2.2.10 Pictorial presentation of results

ccp4mg The CCP4 molecular graphics program. Optimised for the generation of presentation graphics and surfaces.

COOT The Crystallographic Object Orientated Toolkit, for model building, completion and validation.

astexviewer Java program for displaying molecular structures and electron density maps

loggraph Plots graphs of tables from (many) CCP4 programs' log files, as part of ccp4i.

mapslicer Section viewer for CCP4 map files.

O* General model building/manipulation/display graphics program which can produce hardcopy.

molscript* good representation of molecules often used together with raster3D (Kraulis, 1991).

npo Plots maps and draws structure onto them. Various graphical representations.

ortep2* A small molecule drawing program; some of its features are now in npo.

pltdev, xplot84driver, xccpjiffy2idraw Convert PLOT84 metafiles to X-Windows, Tektronix, HPGL or PostScript format. xccpjiffy2idraw converts the result to PostScript which can be edited with idraw.

prepi* a molecular graphics program which can be used to interactively manipulate many different types of molecular representations. (http://bonsai.lif.icnet.uk/people/suhail/prepi.html)

proplot Assorted pretty plots from the procheck suite (q.v.); includes Ramachandran plot.

RasMol X-Windows (and MS Windows) visualiser for PDB files.

raster3D* a set of tools for generating high quality raster images of proteins or other molecules, including rendering pictures composed in molscript.

ribbon* John Priestle's package (Priestle, 1988) for Jane Richardson-type cartoon drawings or graphics displays using frodo. Associated programs: ribrot, postplot, pltout, pdbrot, splitd. (Aggregated.)

setor* hardware-lighted 3-dimensional solid model representations of macromolecules.

topdraw Sketchpad for creating protein topology diagrams.

xloggraph Plots graphs of tables from (many) CCP4 programs' log files under X-Windows.

2.2.11 Utility programs

axissearch Changes axis and cell. (See also `tracer`.)

Babel* Interconverts many coordinate formats.

cad Combine assorted data (and sort) a number of reflection files with various possible operations on the data items. Apart from manipulating the values, data may be converted from one area of reciprocal space to another. Other special functions allow for the generation of input data, for expansion of the data to a lower symmetry if required, and for the generation of data for input to `rsearch`.

cif2mtz program to convert mmCIF structure factors (e.g. from PDB) to MTZ (Martyn Winn).

coordconv Interconverts various coordinate formats.

f2mtz Converts (free-)formatted reflection files to MTZ format.

freerflag Tags each reflection in an MTZ file with a flag for cross-validation.

hklplot Plots "precession" pictures from reflexion files.

hklview is better than `hklplot` if you have X-Windows. (Aggregated.)

matthews_coef Misha Isupov's Jiffy to calculate Matthews coefficient.

mtzMADmod Generates F+/F- or F/D from other for anomalous data.

mtzdump List header and reflections to terminal or printer (Unix script `mtzdmp` runs it more simply.)

mtzmnf Identify missing data entries in an MTZ file and replace with missing number flag (e.g. NaN).

mtztona4 Converts MTZ files to portable NA4 ASCII format. (For exchange with another machine.)

mtzutils Edit columns, title or labels, combine two reflection files. See also `cad`.

mtz2various Produce a file in suitable form for SHELX, X-PLOR, CNS, etc. or in mmCIF or a user-defined format.

na4tomtz Inverse of `mtztona4`.

phistats Performs analysis of agreement between phase sets, and checking against weighting factors.

reindex Reindexes MTZ files when you realise something is wrong. Also change symmetry of residue to asymmetric unit.

sftools Reflection data file utility program including some density map handling

sortmtz Sort and/or merge MTZ files.

stereo Reconstruct 3D coordinates from measurements of stereo diagrams.

tracer Lattice TRAnsformation/CEll Reduction.

wulff.ps For generating Wulff net.

xdldataman Manipulates reflection files, exchanges formats etc. (Kleywegt & Jones, 1996a).

2.2.12 Data Harvesting Utilities

cif2xml Convert mmCIF data harvesting files into XML format.

cross_validate Validate harvesting files prior to deposition.

Data Harvesting Manager A graphical tool to curate data harvesting files, this is part of `ccp4i`.

pdb_extract A suite of programs from the PDB/RCSB for generating harvest files from program logfiles.

2.2.13 Graphical user interface

`ccp4i` is the CCP4 graphical user interface.

2.2.14 Other systems

I hope they give us a plug. . .

BIOMOL The BIOMOL package is a set of programs developed in the Groningen protein crystallography group for most crystallographic tasks which is being converted to use CCP4 file formats.

PHASES Phasing.

Protein "The 'PROTEIN' program system is an integrated collection of crystallographic programs designed for the structure analysis of macromolecules."

Xtal Is a package for small molecule and macromolecular calculations which is available for (reasonably large) PC systems.

XtalView "is a complete package for solving a macromolecular crystal structure by isomorphous replacement, including building the molecular model." X-Windows-based. (Mc Ree, 1993)

Chapter 3. General CCP4 usage

"It's not a modern, iconic, user friendly, menu driven, color, PC compatible user interface," replied the queen, in a tone that would need to come up two notches to be vaguely considered condescending.
"It happens to suit me just fine," retorted Alice.
"What are you an engineer or something?" asked the 7 of spades.
— Alice in DIGITALand *(anon.)*

3.1 Program documentation

Writeups for the programs are currently in HTML one per program and can be viewed through the local Web browser by typing `ccp4help`. Formatted versions are in the $CDOC[1] directory and can be viewed conveniently with an editor.[2] However, the CCP4 installation makes it possible for you to use the man utility, you should be able to use things like `man fft` or `man -k fft` on a Unix system; this is providing the `ccp4.setup` file has been modified correctly (see §11.3).

3.2 Running the programs

Information flows to and from the programs in two ways:

- input and output data files are connected as specified by *command line arguments*, given after the name of the program to be invoked.
- parameters and option specifications are read on the *standard input* stream and a 'log file' of printed output is produced on the *standard output* stream (see §3.4). This input is usually *keyworded* (see §3.2.2).

This mechanism works uniformly under Unix, but *only* if certain system-dependent definitions have been made, usually by executing a command file as part of the login procedure (see §11).

Users of CCP4 Microsoft Windows should use the programs via the graphical user interface `ccp4i`, which hides many of these details (see §3.8).

3.2.1 Command line arguments/file connexion

Input and output data files are specified by associating the with *logical names*. This abstraction was influenced by VMS, with the command line processing following more the Unix model. Logical names are alphanumeric, possibly also including underscores. They are associated with file names of a form appropriate for the operating system, but including a name part and possibly an extension/type, directory specification and version. The association is usually made by specifying name/value pairs on the command line following the program name. Thus the format of a program invocation is

⟨*program name*⟩ [⟨*logical name*⟩ ⟨*file name*⟩] ...

[1] The notation $⟨*name*⟩ refers to the value of a Unix environment variable, defined in the standard startup scripts. In Unix it is substituted using the leading '$'.

[2] The `procheck` documentation is a PostScript file in the `manual` directory as well as a plain text `.doc` file.

where '⟨ ⟩' indicates place holders and '[ ]...' indicates optional repeated items i.e., zero or more, here and elsewhere in the documentation. '|' may also be used to describe syntax, and indicates alternatives between the items on each side.

A concrete example is

```
fft hklin native-Fs.mtz mapout 2Fo-Fc.map
```

Here an FFT is performed on the MTZ reflexion dataset associated with the logical name `hklin` (filename `native-Fs.mtz`) and a map is output to logical name `mapout` (file `2Fo-Fc.map`).

Logical names are case-insensitive - remember that the file names are case-sensitive in Unix, though. Each standard logical name has a default extension associated with it, which is appended is one isn't specified. Thus the `.mtz` and `.map` extensions in the above example are redundant since they are the defaults for the logical names in question.

Unix pitfall: Beware of trying to default the extension for file names which already contain a dot, since the part of the name after the dot will be taken as an explicit extension. (This could arise if, for instance, you were generating filenames with a resolution included as a real number, perhaps in some parameterised script.)

There is a library of standard logical names with associated default extensions and access modes (read, write or read/write). Names from this set which are appropriate to each program are detailed in the programs writeups. Some will default to reading a library file (e.g. of symmetry operators) or writing a file in the scratch directory, and usually are not specified. Otherwise, if a file isn't given for the logical name, the program will attempt to open a file in the current directory with a name corresponding to the logical name.

Some common logical names with default extensions are:

HKLIN input MTZ reflection file (`.mtz`);

HKLOUT output MTZ reflection file (`.mtz`);

MAPIN input map file (`.map`);

MAPOUT output map file (`.map`);

XYZIN input coordinate file (`.brk`);

XYZOUT output coordinate file (`.brk`).

In some cases more than one file corresponding to the same type of entry in `$CINCL/default.def` is required e.g., several input reflexion files. In this case they are typically specified as `HKLIN`⟨*n*⟩, for instance, where ⟨*n*⟩ is a sequence number. The rules for matching logical names with the default specifications given above then apply to the prefix of the name which matches the entries in the files `$CINCL/environ.def` and `$CINCL/default.def`.

In Unix the environment variable `CCP4_OPEN` can be used to determine whether logical names opened as new over-write existing files with the name specified. Over-writing will occur if the value of the variable is `UNKNOWN`; otherwise the program will abort.

For the fine print of the logical name mechanism, see §3.7.

3.2.2 Keyworded input

Most programs take 'keyworded' input to set their parameters. This is read on the standard input in the form of records with a leading keyword followed, possibly, by arguments which might be numbers or strings or keyword/value pairs of the form ⟨*keyword*⟩=⟨*value*⟩. Such arguments are separated by spaces, tabs, commas or = characters. The detail of the input expected is given in the documentation for each program, of course. However, there are some general rules:

- Only the first four characters of keywords are significant (although you are recommended to use complete keywords) and they are case-insensitive;

- Records may be continued across line breaks using `&`, `-` or `\` as the last non-blank, non-comment character on the line to be continued;
- Text following a non-quoted `!` or `#` is treated as a comment and ignored. A continuation character may precede the comment;
- Strings may be single- or double-quoted but the quotes may be omitted if a string doesn't contain whitespace or a comment or continuation character (as above) or if the whole of the rest of the record is read as a single string (which is the case with title information etc.). E.g.:

```
TITLE nothing special
TITLE "don't continue me-"
TITLE '! not a !#" comment'
```

- Leading spaces are ignored
- If an input record is of the form `@`⟨*filename*⟩, with possible surrounding whitespace, the contents of the file ⟨*filename*⟩ will be included as input at that point, after which the following records will be read.

3.2.2.1 Common keywords

Specifications of some keywords common to various programs follow:

`CELL` *a b c* [α β γ]
: Specifies cell dimensions (in Å) and optionally angles in degrees (which default to 90°);

`END`
: Terminate input and start the program going. End-of-file will usually have the same effect;

`GO` | `RUN`
: Are sometimes provided in potentially-interactive programs with the same effect as `END`.

`HEADER NONE` | `BRIEF` | `HIST` | `FULL`
: Controls printing of MTZ information as:
NONE no header output
BRIEF brief header output
HIST brief, with MTZ history
FULL full header output from MTZ reads

`HISTORY` ⟨*history*⟩
: Adds a record to the history stack in an MTZ file, pushing older ones off the bottom if there are too many.

`LABIN` ⟨*program label*⟩=⟨*file label*⟩**...**
: Associates the column labels that the program expects with column labels in the input MTZ file. If there is no ambiguity, the program and file labels can be swapped on the other side of the =.

`LABOUT` ⟨*program label*⟩=⟨*file label*⟩**...**
: Associates column labels in the output file with labels used by the program similarly to `LABIN`.

`RESOLUTION` ⟨*limit*⟩ [⟨*limit*⟩]
: Specifies resolution limits. If only a single ⟨*limit*⟩ is given, it is an upper limit, otherwise the upper and lower limits can be in either order. They are in Å *unless* both are <1.0, in which case they are in units of $4\sin^2\theta/\lambda^2$;

`SYMMETRY` ⟨*number*⟩ | ⟨*name*⟩ | ⟨*operators*⟩
: Specifies symmetry in terms of—e.g. $P2_12_12_1$—either

⟨*number*⟩ spacegroup number e.g. `19`;
⟨*name*⟩ spacegroup name e.g. `P212121`;
⟨*operators*⟩ explicit symmetry operators e.g.

```
X,Y,Z * 1/2-X,-Y,1/2+Z * 1/2+X,1/2-Y,-Z * -X,1/2+Y,1/2-Z.
```

where each follows the conventions of (Hahn, 1992) and matches an entry in the file with logical name `SYMOP` (usually $CLIBD/`syminfo.lib`). Names are the 'short' form given in (Hahn, 1992); the subscripts are typed asis and overbar is typed as a leading `-` so that e.g. $P\bar{1}$ is typed as `P-1`. Explicit operators (rarely needed!) may have the x,y,z triplets separated by spaces or `*` and the coordinate and translation part may be in either order. It is also possible to specify the operators in reciprocal space using h,k,l instead of x,y,z. See also §16.5.1.

`TITLE` ⟨*title*⟩
The rest of the line (or up to 80 characters of it) is taken as a title, usually for an output file.

3.3 Examples

Example scripts to run many programs are available in subdirectories of $CEXAM. Those in $CEXAM/unix/runnable can actually be run using the distributed data in $CEXAM/toxd and $CEXAM/rnase.

This is not a discussion of the details of any particular program so only the script framework is given—the keywords are omitted. The programs are usually run from scripts/ command procedures with the 'keyworded' command data read inline (on the 'standard input' stream) rather than coming from separate files but can be run interactively, of course. One such script/com file often runs several programs in sequence.

Programs can also be run using the graphical user interface `ccp4i` (see §3.8. Entries in `ccp4i` correspond to program keywords. Tutorials using the inteface are distributed in $CEXAM/tutorial.

3.3.1 Unix

The line `#!/bin/csh -fe` in the C-shell example in fig. 3.1 ensures the script is executed by the C shell (without reading `~/.chsrc` and exiting if a command fails). The input is re-directed from the 'here-is document' comprising the lines between the delimiters `EOF-sfall`. Between these delimiters the shell substitutes environment variables preceded by `$` and various other things—see `csh`(1).[3] The exit status of the first program is tested and the second not executed if it fails. The terminating `#` avoids problems if there isn't a trailing newline in the file.

There are some differences if you use the Bourne shell `sh`(1) or a derivative like `ksh` or `bash`, apart from the initial `#!` line (see fig. 3.2). In this case, the `~` metacharacter isn't available (use `$HOME`), the *terminating* word for the << redirection should *not* be quoted if the introducing one is, and it is convenient to chain programs contingent on exit status with `&&`[4] or to use `set -e` at the top of the scrip in modern `sh` implementations. An example of using `sh` and substituting inside the here-is document is the `sortmtz` example in `examples/unix/runnable`.

Which Unix shell you use, at least for scripts, is a somewhat religious issue, but authorities recommend `sh` for various reasons. If you are of the `csh` faith, at least get the improved implementation `tcsh` and use it for writing scripts. See the FAQ for Usenet group comp.unix.shell.

[3] Such substitutions can be useful in re-usable scripts. To turn them off, enclose the delimiters in quotes. This will avoid nasty surprises with shell metacharacters if you don't want the facility.

[4] In `csh` it seems not to be possible to use `&&` with multiple 'here-is' documents in this way.

```
#!/bin/csh -fe
sfall HKLOUT ~/test XYZIN ../Brk/xylose.brk \
      HKLIN xylose << EOF-sfall
⟨your keywords here⟩
EOF-sfall

fft hklin ~/test mapout xylose.map << fft-eof
⟨your keywords here⟩
fft-eof
#
```

Figure 3.1: C-shell script example.

```
#!/bin/sh
sfall HKLOUT $HOME/test XYZIN ../Brk/xylose.brk \
      HKLIN xylose << EOF-sfall &&
⟨your keywords here⟩
EOF-sfall
fft hklin $HOME/test mapout xylose.map << fft-eof
⟨your keywords here⟩
fft-eof
#
```

Figure 3.2: Bourne shell script example.

Unix gotcha: beware of calling an executable script the same name as the program it's running. Depending on your command path, you may end up running the script recursively until you run out of processes or memory. (A good case can be made for avoiding '`.`' in your path and invoking executables in the current directory with a leading '`./`'.)

3.4 Output

The 'log file' output may contain error messages and warnings. If the program encounters a fatal error, it will print a message at the end of the output, along with timing information. If it runs successfully a message like `Normal termination` will be printed with the timing information. The condition code set by the program will indicate success or failure as appropriate for the operating system, e.g. in Unix, zero for success, non-zero on error.

Many of the programs produce graphical information. Output from the programs should be redirected to a 'log file' which can be viewed with `loggraph` (or the older `xloggraph`). Some log files are marked up in HTML, and can therefore be viewed in a browser such as Netscape. Graphs appear as Java applets.

3.5 Rudiments of reflexion files

The MTZ format for reflexion data has some features which it is helpful to understand since CCP4 depends heavily on the standard file formats (or the information the files contain) and the MTZ one in particular. The data are stored in the files (which usually have the 'extension' `.mtz`) notionally as *columns* of real or integer numbers, each with a *label*.[5] The first three columns are the Miller indices of the reflexion, with labels `H`, `K` and `L`. The

[5] The similarity with relational database tables is intentional.

programs expect you to set up a correspondence between the labels in your data file and names of data columns they recognise.

'Standard' MTZ files have one record per reflexion. During the initial data-processing, unmerged data are stored in *multi-record files* in which each reflexion occupies several records which are distinguished by different *batch number*s, symmetry numbers, etc. In this case the fourth column is the 'm/isym' number, recording the symmetry and whether the reflexion is fully recorded, and the fifth column is the *batch number*. Multi-record files have fixed column names which are understood by the relevant programs.

The files also contain various other information in a *header* block. This includes cell parameters, symmetry, column limits, sort order, title and history information. For more details see §16.1. See also §16 for the standard coordinate and map file formats.

3.6 Transporting data files

Reflexion, map and PLOT84 files are binary and are not generally portable between machines with different representations of integer and/or real numbers. However, reflexion and map (but *not* PLOT84[6]) files written using the current CCP4 release are readable on all machines from the set we recognise. They are written using the machine's native number formats and can be read back on the same architecture without overhead. On a different architecture the necessary conversions between foreign and native formats will be done with some overhead (which is not likely to be noticeable on modern machines).

This facility relies on a 'machine stamp' in the file header to identify the architecture on which the file was written. If this doesn't exist the library assumes that the file was written in the native format of the machine with a warning. If you have, say, MTZ files written with an old (pre- release 2.2) version of CCP4, you can easily insert the machine stamp by running `mtztona4` and `na4tomtz` back to back *using the current* `na4tomtz`. Similarly with map files and `maptona4`.

3.7 Details of logical name processing etc.

[*The gory details in this section can be skipped by casual users. They are relevant if you want to redefine or add to the default logical names, for instance.*]

When a CCP4 program starts up, it goes through a *pre-processing* stage before the main body of the program is run (see fig. 3.3). It starts by reading in a file called `environ.def` (in the `include` sub-directory pointed to by `$CINCL` unless you use the `-e` switch—see below). This file specifies the *logical names* that can be expected and some information about the type of file associated with each logical name. A typical line may be `SYMOP=in.lib`, which says that the logical name SYMOP is associated with a file that is input only ('`in`') and has a default extension of `.lib`; other file types are '`out`' (output only) and '`inout`' for read/write files.

After this, the pre-processor then reads in the file `default.def`, which specifies some common logical names defined for all CCP4 processes. A typical line here may be `SYMOP=syminfo.lib` and means that the logical name SYMOP will be associated with the file `syminfo.lib`. You can edit `default.def` to use different files by default e.g., a new atomic scattering dictionary, but usually the defaults are overridden by command line options (§3.7.1).

Finally, the command line arguments are parsed and the logical name/file name pairs are matched up and assigned. An alternative to specifying logical names on the command line is to define a Unix environment variable corresponding to the logical name with the file name as its value, although this is not the recommended way of operating. Command-line

[6] If you need to transport them, use `p842asc` and `asc2p84`.

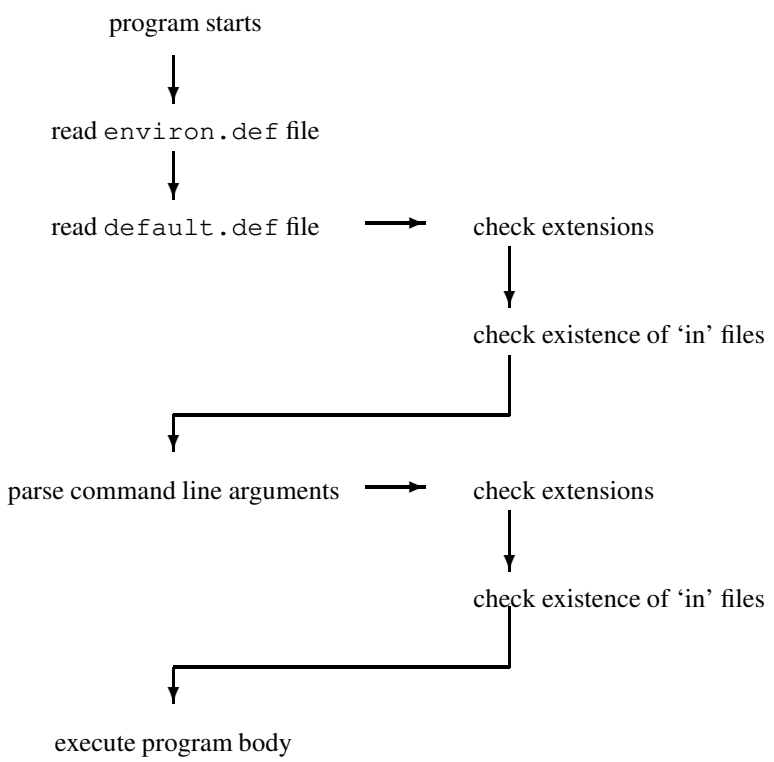


Figure 3.3: Description of CCP4 Program startup

values override any defined in the environment. A logical name on the command line that has not been defined in `environ.def` causes the name to be added to its internal list and a warning to be issued.

Within this procedure are several devices to make running the program easier. Firstly, files without extensions have them appended using the default value, where defined in `environ.def`. Secondly, files with extension `.lib`, `.dic`, `.bes` or `.prt` which do *not* have a full path name have the directory $CLIBD put in front of them (so, logical name SYMOP will actually be associated with $CLIBD`/syminfo.lib`). Thus you only need specify values for logical names such as SYMOP if you want to use other than the library file.[7] Furthermore, files with extension `.scr` are deemed to be scratch files and are assigned to the scratch directory.

Files of type 'in' are checked for existence and the program fails with a message if it can't find them. Under Unix, using output files that already exist will also cause the program to stop unless the environment variable `CCP4_OPEN` is set to `UNKNOWN`, in which case they will be overwritten with a warning message.

[7] N.B. The mechanism for looking for files in $CLIBD can cause confusion if you *do* want to use your own version of a file from the current directory. In this case, either give it an extension different to the list here or use an explicit path e.g,
`SYMOP ./symop.lib` (Unix)

3.7.1 Command line

The programs can take several command line switches to modify the pre-processing that is done, though they are rarely useful. These must be given before the other arguments and have the Unix-style '−' prefix rather than '/' . They are:

–n Do not read the global files `default.def` and `environ.def`;

–v ⟨*0–9*⟩ Verbose output. You can also use `-h` instead of `-v`. Higher values give more debug information from the program, the default value is 1. Use `-v 0` if you want to turn off messages about file opening, otherwise `-v` is currently only useful for debugging. An error occurs if you omit the number;

–d ⟨*file*⟩ Specify an alternative to the `default.def` file;

–e ⟨*file*⟩ Specify an alternative to the `environ.def` file.

3.8 Graphical User Interface

As an alternative to running the programs via scripts as outlined above, a graphical user interface `ccp4i` is also available. The interface is written in the scripting language Tcl/Tk and will run on any platform which supports an interpreter for Tcl/Tk. This includes both Unix and Microsoft Windows platforms.

The main function of the first version of the interface is to make running the programs easier, by hiding many of the details of logical names and keyworded scripts that are outlined above. It also presents options for running the programs in clearer ways than just bare keywords, which makes it particularly accessible for inexperienced users. At the same time, since the programs are independent of the interface more experienced users can jump in and out of `ccp4i` to run one or two programs, without becoming "locked in" to the system.

Within `ccp4i` access to programs is organised by *task*, where a task usually corresponds to one main program but may also use other helper programs. For example, the interface to `fft` may also run the `mapmask` program to extend the map, then use other utilities to convert the map file format to something appropriate for a graphics program. Related groups of tasks are further organised into *modules*, corresponding to different parts of the structure solution process such as Experimental Phasing or Molecular Replacement.

In addition `ccp4i` also has a simple project management system, which allows work to be divided into separate *projects* and which automatically keeps a database[8] of the jobs that have been run in that project. The database retains details of input and output files as well as automatically storing the parameters, which makes it easier to keep track of progress.

The graphical user interface offers some other useful features, for example it has an extensive system of help files in HTML format, and is linked to a number of utilities for viewing different types of files (for example `loggraph`, `sketcher` and `mapslicer`).

To start up the interface within a Unix environment, use the command

```
ccp4i
```

at the command prompt. Within Microsoft Windows environments, click on the `CCP4` icon, or select the appropriate option from the `Start` menu

[8] Though purists will object that this is not a true database.

Chapter 4. Data processing and reduction

"About a thousand and four," said Bruno.
"You mean 'about a thousand'," Sylvie corrected him.
"There's no good saying 'and four': you can't *be sure about the four!"*
— Sylvie and Bruno Concluded

This is a vital part of your experiment. Don't be afraid to repeat it until you are satisfied with the results. Extensive `xloggraph` facilities are available from the programs to monitor the results.

More information on data collection, processing and reduction can be found in the CCP4 Study Weekend proceedings (Helliwell *et al.*, 1987; Sawyer *et al.*, 1993).

4.1 Data processing

By *data processing* we mean that you:

- index the image correctly;
- assign profiles;
- refine orientation etc., to get the best set of *hkl*, I, σI and fractionality where the *hkl* are the measured indices.

Relevant programs are provided, for example, in the `mosflm` system; see also the tutorial information included with it.

4.2 Data reduction

Data reduction is the next stage. We need to use the processed set of reflexions and to produce the optimal set of *hkl*, F and σF, where the *hkl* lie in the chosen unique part of the reciprocal lattice.

The steps involved are:

1. Apply geometric corrections which are independent of data collection device. These are Lorentz and polarisation corrections.
   Try to apply absorption corrections (often not easy but important).
2. Find symmetry equivalents of measured indices.

Steps 1. and 2. may be done by the data processing program or may be in a separate program (e.g. `absurd` for `madnes`).

3. (If you use a data-processing program which doesn't output MTZ files, e.g. `denzo` or `XDS`, use `combat` to produce a multi-record MTZ file from a file of *un-merged* reflexions or `f2mtz` for programs which produce a plain-text file of *merged* reflexions. Some systems can output both merged and un-merged data, so be careful. You may have to extend `combat` for formats it doesn't already support.)
4. Sort(/merge) the data so all equivalent sets of indices are adjacent using `sortmtz`.
5. Use *reliable* equivalent measurements from different sets to calculate relative scale factors. Check the scales make physical sense.
6. Merge equivalent reflexions, applying scale factors.
7. Examine the data carefully for outliers. Try to correct or at least understand as far as possible.

8. Analyse for better estimates of standard deviations, testing data and processing guestimates against observation scatter. Correct guestimates as far as possible.

Steps 5. to 8. are done with `scala` which will need to be run several times.

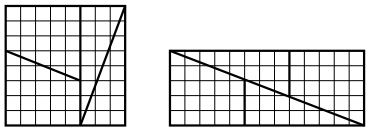
9. Analyse the final data for sensible crystallographic behaviour. Using `truncate`: cumulative intensity distribution; Wilson plot, e.g. with `wilson`.
10. Run uniquify script to complete your dataset, and add freeR column. This should be done for your native dataset. See program `unique` for more information.

To merge data from different sources, you need to produce multi-record files (one from each source, as step 3. above) and go to step 4. iff the files at this stage have the same number and type of columns. (`sortmtz` will merge an arbitrary number of files.) Otherwise use `mtzutils` to merge (sorted) files.[1] To combine low and high resolution data use `scala` with different 'run' numbers for the different resolutions. See the `scala` document for more hints.

To merge unscaled data with a set of previously-scaled data, produce a multi-record file from the scaled MTZ file using `combat` with input type `MTZF` and go to step 4.

[1] Your files may have different columns if you are using, say, `mosflm` and `combat` output together.

Chapter 5. Manipulating and Displaying Data Files



— Graham, Knuth and Patashnik, *Concrete Mathematics*

[More information on data manipulation and display can be found in §2.2.]

5.1 *hkl* manipulation

Reflexion datasets are stored as 'MTZ' files (which are binary). These contain *hkl*, Fs, σs, phases etc. in a labelled column format. (See also §16.1). Utility programs are available to produce and handle these files.

f2mtz reads a formatted file and outputs an MTZ file.

mtz2various exports reflexion files to formats for other systems.

mtzdump dumps part or all of any MTZ file as plain text. You will always get the header information; i.e., column names, types and range. Plus number of zero observations etc.

cad (Collect Assorted Data) is the best program for combining several data sets together. It reads up to 9 input files, with up to 20 assigned columns in each, assigns column types (and new labels if necessary), moves all data to the same asymmetric unit (necessary if you are bringing data together from several sources), and outputs all columns. It can apply a scale factor to all columns in a file (excluding phases). It can also be used to extend your data to cover more of reciprocal space.

sortmtz Sorts/merges MTZ files.

mtzutils has many purposes. It can:

- Select columns from one or two input files;
- Merge (interleave) two sorted files;
- Change header or labels;
- Change symmetry etc.

5.2 Map file utilities and plotting

mama2ccp4 Convert MAMA-format masks to CCP4 format;

mapmask General map and mask manipulation program;

O displays maps (amongst other things);

overlapmap does map manipulation;

npo can contour map sections (amongst other things);

mapdump prints sections of maps and is useful to extract the RMS density deviation for assigning plotting contour levels;

xdlmapman Interconverts formats and does manipulations;

5.3 Interpretation and manipulation of coordinates

Most CCP4 programs use the Brookhaven format for reading and writing coordinates. They require the `CRYST1` and `SCALE`*i* cards to be present and use these to convert to fractional values when required. See also §16.2.

- Manipulation

  bones2pdb Convert `MAMA`-format masks to PDB format;
  coordconv interconverts formats;
  pdbset
  - applies specified rotations and translations;
  - generates symmetry equivalent coordinates;
  - truncates *B*-values outside specified range;
  - renumbers residues, changes chain ids...
  - Inputs/outputs `X-PLOR` format, `O` format etc.
  - finds range of a PDB file.

  lsqkab least-squares fits subsets of coordinates together using the Kabsch algorithm and outputs rotated coordinates;
  baverage averages and analyses *B*-factors residue-by-residue;
  watertidy part of a procedure to move and rename water molecules to sensible values relative to the protein molecule;
  sortwater similar to `watertidy`;
  rwdict converts a PDB file into PROTIN dictionary format;
  chemnotes (commercial) will build you a substrate and output a PDB file.

- Coordinate checking:

  act Various analyses;
  distang calculates distances and angles between near atoms using symmetry. Used to check bad contacts and find hydrogen bonds etc.;
  contact similar;
  procheck checks file for standard naming conventions and sorts the atoms within a residue into the standard order. Checks structural geometry; should be used at all stages of refinement;
  libcheck, refmac5 which are part of the refinement package also check for bad distances, planarity, chirality, torsion angles. They can be used for this purpose without the refinement option.

5.3.1 Converting between fractional and orthogonal coordinates

This is a recurring problem, especially when using coordinates extracted from direct methods programs, or from `mlphare`.

To remind you: If you have a PDB file

$$\begin{bmatrix} x_{\text{frac}} \\ y_{\text{frac}} \\ z_{\text{frac}} \end{bmatrix} = \mathsf{Scalei} \times \begin{bmatrix} X_{\text{orth}} \\ Y_{\text{orth}} \\ Z_{\text{orth}} \\ 1 \end{bmatrix}, \tag{5.1}$$

where

$$\mathsf{Scalei} = \begin{bmatrix} \mathsf{Scale}_{11} & \mathsf{Scale}_{12} & \mathsf{Scale}_{13} & \mathsf{Scale}_{14} \\ \mathsf{Scale}_{21} & \mathsf{Scale}_{22} & \mathsf{Scale}_{23} & \mathsf{Scale}_{24} \\ \mathsf{Scale}_{31} & \mathsf{Scale}_{32} & \mathsf{Scale}_{33} & \mathsf{Scale}_{34} \end{bmatrix}, \tag{5.2}$$

the matrix at the head of a PDB file, i.e.

$$\begin{bmatrix} x_{\text{frac}} \\ y_{\text{frac}} \\ z_{\text{frac}} \end{bmatrix} = \begin{bmatrix} \mathsf{Scale}_{11}X_{\text{orth}} + \mathsf{Scale}_{12}Y_{\text{orth}} + \mathsf{Scale}_{13}Z_{\text{orth}} + \mathsf{Scale}_{14} \\ \mathsf{Scale}_{21}X_{\text{orth}} + \mathsf{Scale}_{22}Y_{\text{orth}} + \mathsf{Scale}_{23}Z_{\text{orth}} + \mathsf{Scale}_{24} \\ \mathsf{Scale}_{31}X_{\text{orth}} + \mathsf{Scale}_{32}Y_{\text{orth}} + \mathsf{Scale}_{33}Z_{\text{orth}} + \mathsf{Scale}_{34} \end{bmatrix}. \tag{5.3}$$

Therefore, extending Scalei to ScaleiExt so that the 4×4 inverse matrix can be generated[1]

$$\mathsf{ScaleiExt} = \begin{bmatrix} \mathsf{Scale}_{11} & \mathsf{Scale}_{12} & \mathsf{Scale}_{13} & \mathsf{Scale}_{14} \\ \mathsf{Scale}_{21} & \mathsf{Scale}_{22} & \mathsf{Scale}_{23} & \mathsf{Scale}_{24} \\ \mathsf{Scale}_{31} & \mathsf{Scale}_{32} & \mathsf{Scale}_{33} & \mathsf{Scale}_{34} \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix}, \tag{5.4}$$

$$\begin{bmatrix} x_{\text{orth}} \\ y_{\text{orth}} \\ z_{\text{orth}} \end{bmatrix} = \mathsf{ScaleiExt}^{-1} \times \begin{bmatrix} x_{\text{frac}} \\ y_{\text{frac}} \\ z_{\text{frac}} \\ 1 \end{bmatrix}. \tag{5.5}$$

The programs `coordconv`, `vectors` and `havecs` will all convert (various formats of) fractional coordinates to orthogonal ones. `havecs`'s `PHARE` input type corresponds to `mlphare`'s output coordinate format.

5.4 Plotting

`mapslicer` can display sections from map files interactively. `npo` can plot map and coordinate files, e.g. for inspecting map sections. It produces a PLOT84 metafile which can be displayed with `xplot84driver` under X-Windows or plotted with `pltdev`. `astexviewer` can display maps and coordinates interactively in three dimensional views. `O` provides general interactive graphics and model building, etc.; there are various programs from Uppsala associated with it. `hklview` and `hklplot` can be used to plot precession pictures. `molscript` and `ribbon` (and others) can plot ribbon diagrams from coordinates.

[1] The extension line ([0.00 0.00 0.00 1.00]) is necessary to cope with $\mathsf{Scale}_{14}, \mathsf{Scale}_{24}, \mathsf{Scale}_{34}$, the translation component of the transformation.

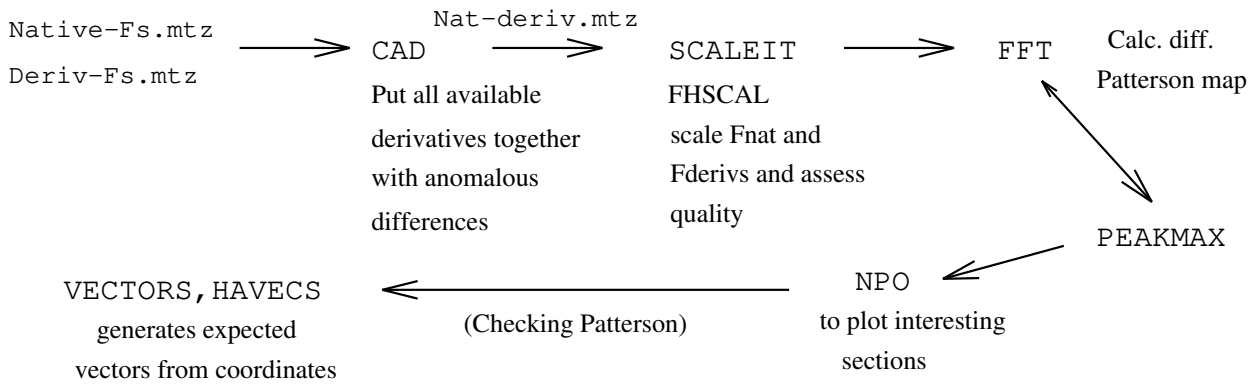
Chapter 6. Isomorphous Replacement

One pill makes you larger
And the other pill makes you small
And the ones that Mother gives you don't do anything at all
— *Grace Slick*, White Rabbit

6.1 MIR structure solutions: steps and programs

1) Find heavy atom sites.

a) Solving Patterson for first derivative to give major site(s):



b) Use Direct Methods to find site(s): SHELX, MULTAN, PATSEE

2)a) Heavy atom refinement: MLPHARE, VECREF

b) Heavy atom phasing: MLPHARE Maximum Likelihood PHAsing and REfinement.

3) Difference Fouriers to find new heavy atom sites in first and other derivatives:

FFT → PEAKMAX → NPO

calculate difference maps

4) Back to calculate and check Pattersons for other derivatives.

Figure 6.1: MIR structure solution.

The main steps in obtaining initial phases by MIR are shown in fig. 6.1 and a tutorial example is available in $CEXAM/tutorial. These steps are:

- Scale derivative data to native using either `scaleit` or `fhscal` (Kraut scaling, probably better). *Be particularly careful about the observations' standard deviations—turn on weighting by them iff you believe what your data processing produces.*
- Find heavy atom sites

1. Solve the Patterson for the first derivative to give the major site(s) (see fig. 6.1 and `rsps`, `vecsum`);
2. Use direct methods to find sites: `SHELX` (`patsee`), `multan`.

Note however that these methods will not in general be able to decide the enantiomorph of the heavy atom site. Also, you must check that all coordinates refer to the same origin.

- Refinement and phasing:
  1. Heavy atom refinement and phasing: `mlphare` (Maximum Likelihood Phasing and Refinement) has several options for this.
  2. Heavy atom refinement: `vecref`. This vector space refinement may be better than `mlphare`'s reciprocal space refinement.
- Difference Fouriers to find new heavy atom sites in first and other derivatives:

  `fft` ⟶ `peakmax` ⟶ `npo`

  with `fft` calculating difference maps.
- Back to calculate and check Pattersons for other derivatives. (It is advisable to do the same procedure *starting* with a different derivative to check for errors or bias; do try to start on the same origin.)

6.2 Combination of partial structure and isomorphous phase information

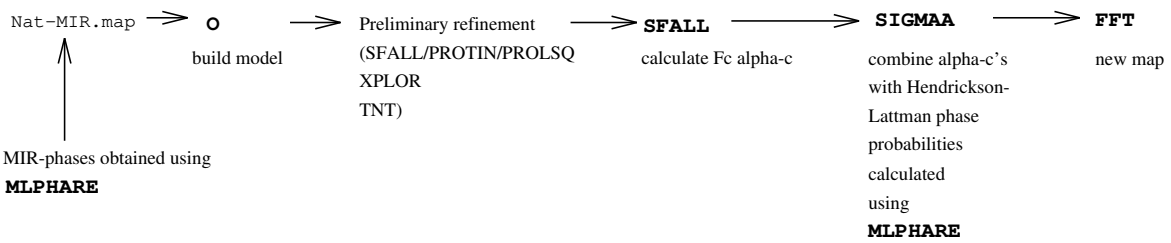


Figure 6.2: Combination of partial structure and isomorphous phase information

6.3 Background

The basic technique of solving the phase problem in protein crystallography is *isomorphous replacement*. Molecular replacement is usually limited to particular cases where a homologous structure is already known. The size of the problem and the intrinsic limitation in resolution make the use of direct methods almost impossible.

Two crystals are said to be *isomorphous* if they have essentially the same structure but are composed of chemically different atoms. In protein crystallography what is called 'isomorphous replacement' should properly be named 'isomorphous addition', since it is usually the binding of a heavy atom in a position previously occupied by disordered solvent. The electron density of the isomorphous derivative should differ from the electron density of the native crystal just for the peaks due to one or a few specifically-bound heavy atoms; the unit cell dimensions and the protein molecule must be essentially the same in both.

How heavy should the heavy atom be to cause useful changes in the diffraction pattern? Perutz (Green *et al.*, 1954) first pointed out that, although one might doubt that the presence of a couple of Hg atoms would produce significant intensity changes in the diffraction

pattern generated by 5000 C, N and O atoms, many of the scattering contributions of the light atoms cancel out by interference, while 80 Hg electrons scatter in phase.

In a more rigorous treatment Crick and Magdoff (1956) compared the average change in intensity produced by adding extra atoms to the crystal with the average change in intensity produced by small shifts of the protein molecules and slight changes in the unit cell dimension. They showed that the r.m.s. fractional change in intensity I produced by the stoichiometric binding of $N_{\rm H}$ heavy atoms of scattering power $f_{\rm H}$ to a protein of $N_{\rm P}$ atoms having a mean scattering power $f_{\rm P}$ is

$$\langle \Delta I^2 \rangle^{1/2} / \langle I \rangle \simeq (N_{\rm H}/N_{\rm P})^{1/2} (f_{\rm H}/f_{\rm P}). \tag{6.1}$$

This analysis is useful not only to give a clue about the size of the heavy atom required, but also to check how flexible the definition of isomorphism should be. Their conclusion was that small shifts either of the molecule or of the cell dimensions may produce changes in the intensity sufficient to interfere seriously with the changes due to the heavy atoms at high values of $1/d$. They also proposed some methods to identify lack of isomorphism. One could examine the variation of $\langle \Delta I^2 \rangle / \langle I \rangle$, or alternatively of $|\overline{F_{\rm PH} - F_{\rm P}}| / \overline{F}_{\rm P}$, with resolution: if the ratio increases at high $1/d$ some movements of the protein have taken place.

The 'Normal' analysis (Smith & Howell, 1992) is a good method of assessing a derivative *if* the standard deviations are reasonable. They suggest looking at the distribution of isomorphous differences against standard deviations for all the observations. The central limit theorem predicts that if the $F_{\rm ph}$ measurements are essentially the same as the $F_{\rm p}$ measurements 66% of the differences should lie within one standard deviation of the mean. If this is not so, it is possible that there is a real difference between the two sets of data.

Serious problems in the search for heavy atom derivatives can arise in the case of crystals with very large asymmetric units. From Crick and Magdoff's analysis it is clear that, in order to have a detectable difference in intensity, when $N_{\rm P}$ increases either $N_{\rm H}$ or $f_{\rm H}$ must increase. Obviously $f_{\rm H}$ is limited by the number of electrons of the heavier stable atoms available. It is possible to label the protein at many sites by standard heavy atom compounds, but derivatives containing single atoms at multiple sites are usually difficult to solve and the risk of obtaining non-isomorphous crystals is increased. What is needed is the scattering power of multiple substitutions concentrated at one or a few sites; this can be achieved by the use of clusters. In fact, for a derivative with one heavy metal at $N_{\rm H}$ sites $\langle \Delta I^2 \rangle^{1/2} / \langle I \rangle \simeq (N_{\rm H}/N_{\rm P})^{1/2} f_{\rm H}/f_{\rm P}$, while for a complex of $N_{\rm H}$ atoms in a single site $\langle \Delta I^2 \rangle^{1/2} / \langle I \rangle \simeq (1/N_{\rm P})^{1/2} N_{\rm H} f_{\rm H}/f_{\rm P}$. The advantage of using groups in contrast to conventional compounds becomes especially evident when the sites of substitutions are to be located in a difference Patterson.

More information on isomorphous replacement can be found in the Study Weekend proceedings (Wolf *et al.*, 1991) and Blundell and Johnson (1976). Useful references to the development of the method are: Green *et al.* (1954), Harker (1956), Perutz (1956), Crick and Magdoff (1956), Blow (1958), Kendrew *et al.* (1958) and Blow and Crick (1959).

6.4 Determination of heavy atom positions

In order to use the isomorphous replacement method, the amplitude and phase of the heavy atom contribution to the structure factor of the derivative has to be determined. This means that the sites of substitution must be worked out from the observed differences in scattering amplitudes caused by the introduction of the heavy atoms. The techniques used to do this are:

- Patterson searches of different kinds;

- direct methods;
- difference Fouriers.

These have to work with some estimate of F_H obtained only from the observed differences.

6.4.1 Scaling between native and derivative data

The first step is to scale the native and derivative data sets together. In comparing different sets of data the normal procedure is to use a Wilson plot to put both of them onto the same absolute scale but this is never very reliable with data to a resolution below 2.5 Å. A sort of relative Wilson plot, comparing $\langle F_P^2 \rangle$ and $\langle F_{PH}^2 \rangle$, can also be used (Dodson, 1976). This will not be accurate since no allowance can be made for the additional contribution from $\sum f_H^2$, and this quantity depends on the degree of heavy atom substitution, which is difficult to estimate at the beginning. However, the various Patterson functions are not very sensitive to errors in scale of up to 10%, and this estimate is usually good enough for preliminary calculations.

Kraut suggested a relatively simple way of calculating the derivative scale factor by equating the Patterson origins:

$$\sum F_H^2 = \sum (K_s F_{PH})^2 - \sum F_P^2 \tag{6.2}$$

where K_s is the scale factor to be estimated (Kraut *et al.*, 1962; Tickle, 1991); see also the `fhscal` documentation. This method often gives excellent results.

Once the heavy atom positions have been determined the relative scale and temperature factor can be refined along with the other heavy atom parameters (positional coordinates and occupancies).

6.4.2 Estimation of F_H

It is necessary to estimate the amplitude of the heavy atom contribution F_H. Once this is done we can sum the contributions to give a Patterson map where we should see heavy atom–heavy atom vectors. The structure factors of the native protein $\mathbf{F}_P$, the derivative $\mathbf{F}_{PH}$ and the heavy atom structure $\mathbf{F}_H$ are related by the vector equation:

$$\mathbf{F}_H = \mathbf{F}_{PH} - \mathbf{F}_P. \tag{6.3}$$

We can only get the magnitudes of $\mathbf{F}_{PH}$ (F_{PH}) and of $\mathbf{F}_P$ (F_P) from experiment. As the phase information is unavailable at this stage it is impossible to use the equation above to calculate F_H. Therefore we must estimate it using the procedures defined below.

6.4.2.1 Centric reflexions

For centric reflexions $\mathbf{F}_{PH}$ and $\mathbf{F}_P$ are collinear. This means F_H must equal either $|F_{PH} - F_P|$ or $|F_{PH} + F_P|$. In general the contribution of the heavy atom is small with respect to the structure factor of the protein and the phases of $\mathbf{F}_{PH}$ and $\mathbf{F}_P$ are very likely to be the same. Therefore F_H should be equal to $|F_{PH} - F_P|$. In a few cases, usually with very weak reflexions, the sign of $\mathbf{F}_{PH}$ can be different from the sign of $\mathbf{F}_P$ and F_H should be evaluated by $|F_{PH} + F_P|$ (*crossing-over*).

6.4.2.2 Acentric reflexions

For acentric reflexions the phases of $\mathbf{F}_{PH}$ and $\mathbf{F}_P$ will not, in general, be correlated and therefore the exact relationship would be given by:

$$\begin{aligned} F_{iso} &= |F_{PH} - F_P| \\ &= \left| F_H \cos(\alpha_{PH} - \alpha_H) - 2F_P \sin^2 \frac{(\alpha_P - \alpha_{PH})}{2} \right| \end{aligned} \tag{6.4}$$

and, since usually F_H is rather small compared to F_P and F_{PH}, we can assume $\alpha_P \simeq \alpha_{PH}$ and $\sin^2(\alpha_P - \alpha_{PH}) \simeq 0$. It follows that

$$|F_{PH} - F_P| \simeq |F_H \cos(\alpha_{PH} - \alpha_H)|. \tag{6.5}$$

This term is usually adequate for calculating an isomorphous difference Patterson (see below).

Anomalous scattering data can also be used to estimate F_H. The approximate formula is:

$$\begin{aligned} \Delta F_{ano} &= |F^+_{PH} - F^-_{PH}| \\ &\simeq 2k^{-1} F_H \sin(\alpha_{PH} - \alpha_H) \end{aligned} \tag{6.6}$$

where k is defined as F_H/F''_H and can be obtained from the International Tables although it is more usual to substitute an empirical value derived from the data (see below).

Although both isomorphous and anomalous data are inaccurate, they give complementary information and used together can give a better estimate for F_H. The problem arises of how isomorphous and anomalous data may be put together to give an accurate estimate of F_H. The simplest way is to *sum* the contributions with the appropriate scaling:

$$\Delta F^2_{iso} + (k^2/4)\Delta F^2_{ano} \simeq F^2_H \cos^2(\alpha_{PH} - \alpha_H) + F^2_H \sin^2(\alpha_{PH} - \alpha_H) = F^2_H. \tag{6.7}$$

A more precise equivalent expression can be used to *combine* the isomorphous and anomalous data (Matthews, 1966; Singh & Ramaseshan, 1966):

$$F^2_H \simeq F^2_M + F^2_P \pm 2\left[F^2_M F^2_P - (k/4)^2(\Delta I)^2\right]^{1/2}, \tag{6.8}$$

where $F^2_M = \frac{1}{2}(F^{+\,2}_{PH} + F^{-\,2}_{PH})$ and $\Delta I = F^{+\,2}_{PH} - F^{-\,2}_{PH}$ and we assume $1/k^2 \ll 1$.

As the isomorphous differences are only small, only the solution with the negative sign is normally considered, yielding F_{HLE} (Heavy atom Lower Estimate)

$$F^2_{HLE} = F^2_M + F^2_P - 2\left[F^2_M F^2_P - (k/4)^2(\Delta I)^2\right]^{1/2}. \tag{6.9}$$

k, the ratio of the real and imaginary parts of the heavy atom structure factor can be approximated from the data (Matthews, 1966):

$$k = 2\overline{|F_{PH} - F_P|} \Big/ \overline{|F^+_{PH} - F^-_{PH}|} = 2\overline{\Delta F_{iso}} \Big/ \overline{\Delta F_{ano}}\,. \tag{6.10}$$

The anomalous differences are usually overestimated, and this causes an overestimation of F_{HLE} for acentric reflexions. French (Dodson *et al.*, 1975) suggested a formula for the bias for F_{HKL} and proposed the use of $(F_{HKL} - \text{bias}(F_{HKL}))^2$ to calculate Pattersons.

Isomorphous and anomalous data (as well as any prior information) can be combined correctly in a maximum likelihood approach (Bricogne, 1991). `mlphare` provides a rough approximation to the correct approach.

6.4.3 Determination of the heavy atom positions

The estimated F_Hs are used to find the heavy atom positions. These coordinates will then be used to calculate the vector $\mathbf{F}_H$ required to find the protein phases.

6.4.3.1 Patterson methods

The most general method that does not require any previous information is the use of Patterson techniques. An isomorphous difference Patterson synthesis is a Fourier summation of the terms

$$|F_{\rm PH} - F_{\rm P}|^2 \simeq |F_{\rm H}\cos(\alpha_{\rm PH} - \alpha_{\rm H})|^2. \tag{6.11}$$

In the case of centric reflexions $|F_{\rm PH} - F_{\rm P}|$ really corresponds to $F_{\rm H}$ and a Patterson with $(F_{\rm PH} - F_{\rm P})^2$ coefficients is a good estimate of a $F_{\rm H}^2$ Patterson; the only noise is due to the few cases of crossing-over, and to measurement errors. However, erroneously large differences and the lack of centric data can reduce the quality of the Patterson.

For acentric reflexions, we can use

$$\cos\theta^2 = (1 + 2\cos(2\theta))/2 \tag{6.12}$$

to show that the summation has two terms: ${F_{\rm H}}^2/2$ and ${F_{\rm H}}^2\cos 2(\alpha_{\rm PH} - \alpha_{\rm H})$. The first term will generate vectors between the heavy atoms at half the ideal height. The second term will contribute only noise, since there is no connection between the $F_{\rm H}$ and the phases used.

The fact that for acentric reflexions the magnitude of the differences between native and derivative amplitudes is not generally equal to the heavy atom structure factors does not represent the only cause of noise. Much more serious sources of noise are measurement and scaling errors, as well as lack of isomorphism. Since the size of the $F_{\rm H}$ contribution is two or three percent of $F_{\rm PH}$ and $F_{\rm P}$, it may be of the same order as the standard deviation of the observations, and since a Patterson function is dominated by its largest terms, a few spuriously large differences can disguise the Patterson signal completely.

An anomalous difference Patterson synthesis is a Fourier summation of the terms

$$|F_{\rm PH}^{+} - F_{\rm PH}^{-}|^2 \simeq 2k^{-1}F_{\rm H}\sin(\alpha_{\rm PH} - \alpha_{\rm H})^2 \tag{6.13}$$

The formula $\sin\theta^2 = (1 - 2\cos(2\theta))/2$ means the summation of $\Delta F_{\rm ano}^2$ should also give peaks in the Patterson corresponding to vectors between heavy atoms plus noise. There is no lack of isomorphism in the anomalous difference, but in general the F'' signal is much weaker than the $F_{\rm H}$ signal, so the random errors inherent in the measurements can swamp it more easily.

Better Patterson maps can sometimes be obtained using estimates of $F_{\rm H}$ which take into account both isomorphous and anomalous data i.e. the *summed* or *combined* ($F_{\rm HLE}$) Patterson. However if the measurements are reasonably accurate, and the heavy atom is well substituted, both the isomorphous and anomalous difference Pattersons should be interpretable.

6.4.3.2 Solving the Patterson

Even an error free Patterson generated from several sites can be hard to interpret, especially in a high symmetry space group. Automated search procedures for locating heavy atoms in difference Patterson maps have also been developed (Terwilliger & Eisenberg, 1987); see also `rsps` and `vecsum`. A systematic search procedure is especially necessary in the presence of high non-crystallographic symmetry, as in crystalline viruses (Rossmann *et al.*, 1986). These automatic procedures can fail when two vectors that are close overlap to form one big peak. Thus the heavy atom may be placed incorrectly at a special position. Solution by hand may be the only way of avoiding this. It is good practice to find consistency between Pattersons, calculated from different resolution ranges, rather than to rely on just one. Difference Pattersons can determine whether your solution is correct, see below.

6.4.3.3 Direct methods

The location of a larger number of heavy atoms by difference Patterson becomes increasingly difficult and the use of direct methods can help in solving the problem.

Standard direct method programs (for example SHELX86 (Sheldrick, 1991) or MULTAN (Germain *et al.*, 1970)) can sometimes find heavy atom sites consistent with the isomorphous or anomalous differences (Mukherjee *et al.*, 1989).

6.4.3.4 Difference Fouriers

When a previous estimate of the protein phases exist (for example when we are dealing with the n^{th} derivative, or looking for further sites in the current derivative) a Fourier synthesis with coefficients $m(F_{\text{PH}} - F_{\text{P}})e^{i\alpha_{\text{P}}}$, where m is the weight associated with this phase, is the simplest way to locate the heavy atoms. This method usually works even when the protein phases are very inaccurate.

The biggest problem with this method is the fact that these Fourier syntheses will often have 'ghost' peaks or halos at or near the heavy atom sites used for phasing. These can be erroneously interpreted as minor sites. It is sensible to check any sites against the derivative Pattersons.

6.4.3.5 Choice of origin and hand

Any Patterson function will be equally well satisfied by coordinates x, y, z or $-x, -y, -z$. This is true for direct methods techniques as well. In other words, the *hand* of the solution will not be determined. In addition, in many spacegroups there is an arbitrary origin choice, and it is essential that all derivatives use the same origin and are on the same hand. The first choice is in fact arbitrary, but once fixed—for example assigning the coordinates to an heavy atom in one isomorph—must be used consistently for all subsequent derivatives. The use of a difference Fourier based on the phases of the first derivative will give the sites of the subsequent derivatives on the same relative origin and hand (Matthews, 1966). Various correlation functions working in Patterson space have also been used. Perutz (1956) proposed two correlation functions, later improved and modified (Blow, 1958). Rossmann (1961) suggested a method based on the calculation of difference Pattersons between pairs of derivatives, with coefficients $(F_{\text{PH}_1} - F_{\text{PH}_2})^2$. This synthesis will give positive peaks for vectors between atoms in the same derivative, and negative peaks for the correlation vector between atoms in different isomorphs. More elaborate correlation functions have been proposed (Kartha & Partharasarathy, 1965).

6.5 Refinement of heavy atom parameters

After rough heavy atom parameters have been found they must be refined to their best values before being used for phase determination. An important aim of the refinement is to discover all the possible minor sites. The techniques are similar to the ones used to refine small molecule structures but there are more difficulties due to the fact that, especially for acentric reflexions, the estimate of F_{H} can be relatively poor and errors due to lack of isomorphism and scaling cause the data to be less accurate. Usually, since the ratio observations/parameters to be refined is high, it is wise to reject any suspect reflexions, as long as those remaining are uniformly distributed through $\sin^2\theta$.

6.5.1 "Maximum Likelihood Phase" refinement

In 'Maximum Likelihood phase' refinement (Otwinowski, 1991) we find a probability distribution for values of the protein phases ϕ_{P} and then minimise the weighted differences between observed and calculated values of F_{PH} generated with this phase, with respect to the heavy atom parameters. These are:

- relative scale and temperature factor between protein and derivative data;
- coordinates;
- real and anomalous relative occupancies;
- thermal factors (either isotropic or anisotropic);
- error estimates which are used when finding the appropriate weight for each observation.

The parameters are updated and another cycle of refinement is carried out. The function to be minimised is:

$$R_j = \sum_{\phi_P} \sum_{\mathbf{h}} w(\phi_P) \left(F^{\text{obs}}_{\text{PH}_j} - F^{\text{calc}}_{\text{PH}_j}(\phi_P) \right)^2 \tag{6.14}$$

where j refers to the j^{th} derivative.

This procedure refines scale factors, relative occupancies and estimated errors for different derivatives. It is not seriously biased by incorrect models. Since all possible protein phases are sampled for each reflection, the procedure rapidly eliminates incorrect sites. For reflections with well defined phases the weight of the protein phases will be very small for all values except the best phase. But there are many reflections where the protein phase is poorly defined and for these reflections the 'maximum likelihood' approach avoids the old problem of biasing the results towards phases derived from incorrect sites.

Note that `mlphare` does not implement a correct approach to maximum likelihood estimation, which should involve a 'straightforward' optimisation of the likelihood function after integrating out the unknown native F and phase (Bricogne, 1991; Read, 1991). This optimisation is carried out at least as a function of the (resolution-dependent) lack of closure (non-isomorphism) parameters and possibly also as a function of the heavy atom parameters, although vector space refinement (see below) may be a better way of determining those if anomalous data are not involved.

This refinement can be quite slow, and if there are sufficient centric data it is sensible to refine against them first. When there are two independent derivatives it is possible to use the sign of the 'anomalous occupancy' to decide the hand of the solution. If the anomalous occupancies are negative, the heavy atom sites are very likely to lie at $-x,-y,-z$. (Another possibility is that you have confused the intensity observations and indexed the (h,k,l) as $(-h,-k,-l)$.)

The final and essential check is to calculate difference Fouriers (or "residual maps") with coefficients

$$\sum_{\mathbf{h}} (\Delta F_{\text{iso}} - F^{\text{calc}}_{\text{H}}) e^{i\alpha_{\text{H}}} \tag{6.15}$$

These may show some peaks corresponding to new heavy atom sites.

6.5.2 Vector space refinement

Vector space refinement as implemented in `vecref` has advantages over the reciprocal space treatment in `mlphare`. It is not known to have been tried with anomalous (MAD) data but should work. See (Tickle, 1991).

6.5.3 "F_{H}" refinement

In this scheme the function to be minimised is a difference between the "observed" and calculated F_{H}. It was first proposed (Rossmann, 1960) as a procedure to refine heavy atoms parameters for two derivatives. A modified version of F_{HLE} refinement has been proposed (Terwilliger & Eisenberg, 1983). This was implemented in the program `heavy` but has been superseded by the maximum likelihood technique of `mlphare`.

6.6 Phase determination

6.6.1 Isomorphous replacement data

From the vector triangle $\mathbf{F}_{PH} = \mathbf{F}_P + \mathbf{F}_H$, knowledge of the amplitudes F_{PH}, F_P, F_H, the phase α_H and using the cosine law, α_P can be derived:

$$\begin{aligned} \alpha_P &= \alpha_H + \cos^{-1}((F_{PH}^2 - F_P^2 - F_H^2)/2F_PF_H) \\ &= \alpha_H \pm \phi \end{aligned} \tag{6.16}$$

To obtain an unambiguous estimate of α_P we need to repeat the phase determination using another vector $\mathbf{F}_{H_2}$, not collinear with the first one, since the two solutions are symmetric with respect to α_H; this can be done by using another derivative with heavy atoms substituted at different sites.

Harker (1956) gave an elegant geometrical solution to the problem of phase determination: he showed that if a phase circle of radius F_P is drawn from the origin of the Argand plane, and another circle of radius F_{PH_1} is drawn from the end of the vector $-\mathbf{F}_{H_1}$, their points of intersection will give the two solutions for α_P. With a second derivative we obtain a circle with radius F_{PH_2} and origin at $-\mathbf{F}_{H_2}$, that should intersect the F_P circle at one of the two previous values, giving a unique solution for α_P.

In practice, however, experimental errors, approximations in determining $\mathbf{F}_H$ and lack of isomorphism affect the precision of this graphical construction and usually we need more than two derivatives to obtain satisfactory protein phases—MIR or Multiple Isomorphous Replacement.

6.6.2 Anomalous scattering data

In phase determination using anomalous data, the out-of-phase component $\mathbf{F}''_H$ acts in the same way as the entire vector $\mathbf{F}_H$ in the isomorphous method; we can obtain $\mathbf{F}^+_{PH}$ from $\mathbf{F}^-_{PH} + 2\mathbf{F}''_H$ in the same way as $\mathbf{F}_{PH}$ from $\mathbf{F}_P + \mathbf{F}_H$. Since $\mathbf{F}''_H$ and $\mathbf{F}'_H$ are in general perpendicular one each other, they give complementary information and, in the absence of errors, one could get a unique solution for the phase α_P by combining isomorphous and anomalous data. Although this phase solution is unique, it is essential that phases are generated from both possible heavy atom solutions (x, y, z) and $(-x, -y, -z)$. Each solution will give the same phasing statistics, one will be approximately correct, and the other wrong. The solutions obtained by combining the Single Isomorphous Replacement and Anomalous Scattering data using a single derivative are referred to as SIRAS phases. Phasing by SIRAS benefits from the use of synchrotron radiation, where it is possible to select the wavelength to optimise the anomalous contribution. It is possible to obtain the hand of the heavy atoms from one anomalous derivative at one wavelength, (Woolfon & Yao, 1994)

6.6.3 Multi-wavelength Anomalous scattering Data

The values of f' and f'' are wavelength dependent. Thus if data are collected at several wavelengths, there will be small 'isomorphous' changes between the F_{hkl}s, and the anomalous differences will also have different magnitudes. The advantage of this is that only one derivative needs to used and the data sets are all isomorphous, if they are from the same crystal. The disadvantages are that one needs a tunable X-ray source and the experiments are more complex. The increased complexity is due to the fact that the effective signals are even smaller than normally present in MIR or SIRAS. The signal and thus the wavelengths need to be optimised. This is achieved by using wavelengths around an absorption edge for the heavy atom in question. See Hendrickson (1991) for more details on experimental procedures. These differences can be used to determine the protein phase, either using

`madsys` or by maximum likelihood phasing (`mlphare`); the latter is probably the method of choice.

Although there is no 'native' in a MAD experiment, `mlphare` requires an effective native to be defined. This can lead to negative occupancies. To avoid this, use the data set with the largest negative value for f' as the 'native'. Negative occupancies are not necessarily incorrect and do not affect the phasing. It just means that the average scattering power of the 'derivative' is less than that of the 'native'.

6.6.4 Density modification

Isomorphous phases are usually improved by use of a density modification procedure (§8) before the initial map is calculated.

6.6.5 Treatment of errors

[*Errors are now refined as part of the Maximum Likelihood refinement and phasing procedure.*]

Up to now we have assumed that the analysis is free of errors. In practice, however, there are errors that can be thought of as deriving from two different sources:

- Errors in measuring intensities of reflexions will produce errors (δ) in the amplitudes $F_{\rm P}$ and $F_{\rm PH}$;
- Errors in locating and describing the scattering of the heavy atoms as well as lack of isomorphism will introduce errors (ε) in $\mathbf{F}_{\rm H}$.

The result is that the two derivative circles will not intersect the native one at precisely the same point. The total error E, is a measure of the discrepancies between theory and experiment:

$$\langle E\rangle^2 = \langle\delta\rangle^2 + \langle\varepsilon\rangle^2, \tag{6.17}$$

where $\langle\delta\rangle$ can be estimated by comparing symmetry equivalent reflexions from the same crystal or the same reflexions from different crystals. The estimation of $\langle E\rangle$ can be made for centric reflexions from the difference between $F_{\rm PH} - F_{\rm P}$ and $F_{\rm H}$: in acentric cases this is not possible because $F_{\rm PH} - F_{\rm P}$ is not a correct estimate of $F_{\rm H}$. The error estimate is refined during the maximum likelihood phase refinement.

The influence of such errors was originally discussed by Blow and Crick (1959). See also Blow and Rossmann (1961), Bricogne (1991), Otwinowski (1991) and Read (1991).

One hopes that the use of more than one derivative will solve the phase ambiguity; unfortunately it happens that in many cases there is a strong tendency for the probability distribution of the phases to be bimodal and a strategy for making the "best" choice needs to be developed. In fact, the obvious choice of taking the maximum of the probability distribution can lead to big errors where other choices have a certain probability of being correct. We need a compromise between these different choices, in such a way as to minimise the mean squared error in the electron density. This "best" Fourier is calculated with phases corresponding to the centroid of the distribution weighted by the *figure of merit*, m, which is a measure of the reliability of the phase determination. If the probability is sharp and unimodal m will be close to 1, but if the probability is almost uniformly distributed along the circle, m will be very small.

Therefore, the best Fourier has coefficients

$$mF_{\rm P}e^{i\alpha_{\rm best}}. \tag{6.18}$$

It can be shown (Dickerson *et al.*, 1961) that m is related to the mean value of the cosine of the error in the phase angle for the reflexion:

$$m = \langle\cos\Delta\alpha_i\rangle \tag{6.19}$$

(for example $m = 0.866$ corresponds to an expected error of 30° in α_P).

The maximum likelihood approach of assigning each possible phase a likelihood and combining these means that weights are more realistic (i.e., lower) and phases more accurate since spurious sites are lost more completely.

An alternative procedure avoiding the limiting assumption that all the errors lie in the determination of F_{PH} has been proposed (Cullis *et al.*, 1961).

6.6.6 Hendrickson–Lattman coefficients

The formulation of the phase probability function given by Blow and Crick has the disadvantage that for every reflexion a summation must be carried out over all the derivatives. This means that every time new derivatives (or phase information from other sources) are available, everything has to be re-calculated. A variation of the Blow and Crick method (Hendrickson & Lattman, 1970) represents the phase probability for each derivative by the equation:

$$P_i(\alpha) = e^{k+A\cos\alpha+B\sin\alpha+C\cos 2\alpha+D\sin 2\alpha}. \tag{6.20}$$

Calculating the probability function using various information is simply a matter of summing the corresponding coefficients.

6.6.7 Single Isomorphous Replacement

The phase ambiguity can be removed by another isomorphous compound, but the need for two or more derivatives has often created difficulties, since their preparation is one of the most time-consuming steps in protein crystallography. Therefore it is of interest to minimise the number of isomorphous crystals which are needed. In some cases just one derivative can be enough.

The best results can be obtained if data from anomalous scattering are available but there are limitations due to the fact that anomalous differences are small and would require the diffraction data to be measured with a degree of accuracy that is often unobtainable; SIRAS phases are normally considered the first step in an MIR determination, instead of the final result.

A different approach that utilises a procedure known as noise filtering has been proposed by Wang (see §8). Analogous procedures which make use of different physical constraints have been used to improve the electron density maps; they have been reviewed (Podjarny *et al.*, 1987). However, all these approaches are more useful for 'cleaning' a bad MIR map than actually solving phase ambiguities in SIR phases.

There are a number of techniques used to break the phase ambiguity of one-wavelength anomalous scatterering with some success (Ralph & Woolfson, 1991; Fan *et al.*, 1990; Hao & Woolfson, 1989).

Chapter 7. Molecular Replacement

"He must have imitated someone else's hand," said the King. (The jury all brightened up again.)
— Alice's Adventures in Wonderland

7.1 Molecular replacement: steps and programs

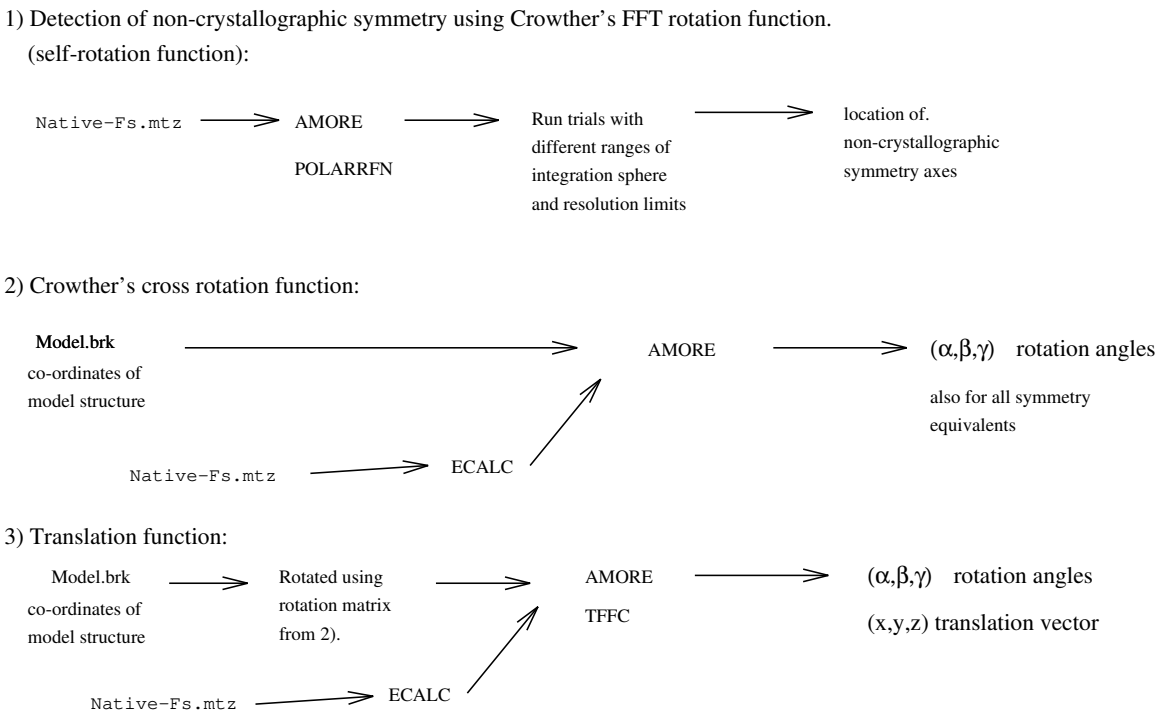


Figure 7.1: Molecular Replacement

The main steps in obtaining initial phases by Molecular Replacement are shown in fig. 7.1. The principal program is `amore` (Navaza, 1994) which is a state-of-the-art molecular replacement program incorporating the rotation and translation steps plus rigid body fitting within one program[1].

[1] In the CCP4 implementation—this has diverged from Navaza's.

7.1.1 Detection of non-crystallographic symmetry using self rotation function

`amore` or `polarrfn` yields the rotation function from native Fs. Run trials with different ranges of the integration sphere and resolution limits to locate non-crystallographic symmetry axes. (`amore`'s rotation function is superior to `almn`'s (Navaza, 1993; Driessen & Tickle, 1994).)

7.1.2 Cross rotation function

`pdbset` finds the dimensions of a model molecule from the input PDB file.

`sfall` calculates structure factors in a P1 cell for the model if necessary (not necessarily for `amore`); `ecalc` may be used to produce normalised structure factors (Es), which may give better results (also for the translation function).

`amore` uses the the externally- or internally-calculated Fs (or, preferably, Es calculated externally) and selected radii and resolution ranges to evaluate the rotation function. This is described by the Eulerian angles (α, β, γ); all symmetry equivalents are listed. NB. the rotation function in `almn` is inferior to that in `amore`.

`amore`, `CNS`, `MERLOT` calculate the cross rotation function.

7.1.3 Translation function

`pdbset` rotates the coordinates of the model structure and outputs a new set of coordinates.

`sfall` takes the resulting coordinates and calculates structure factors for this model in the new cell without applying the symmetry operators.

`cad` collects $F_{\rm obs}$s and $F_{\rm c}$s for all symmetry equivalents.

`tffc` calculates translation function Fourier coefficients. A phased translation function may also be calculated using `fft`. `rsearch` may also be used in particular cases where you know the solution must be somewhere special but this method is computationally inferior.

`fft` then calculates the map.

`amore's` translation function may be run for many possible RF solutions at once, but is somewhat inferior to `tffc`s at present.

`mapsig` prints TF map statistics.

7.1.4 Checking the results

`distang` checks for bad symmetry contacts. There are various other programs for checking packing, including ones from `merlot` and `pacman` from Uppsala.

graphics

rigid body refinement e.g., `amore` (Castellano *et al.*, n.d.), `CNS`

7.2 Introduction

It is often possible to determine the structure of a molecule using a similar or homologous structure that is already known. One such case may be when we try to determine the crystal structure of a known molecule which has crystallised in a different space group. Another area in which *molecular replacement* (MR) is particularly useful is site-directed mutagenesis where large numbers of mutants are produced and studied. Even a minor change in sequence may change the way a protein crystallises. The most economical way to solve such structures is usually MR. Another of its uses involves different proteins belonging to the same structural family. As the number of known structures increases it is becoming apparent that the number of distinct protein folds is limited and that even functionally unrelated proteins may share a common fold. It is therefore increasingly likely that a new protein has a structure related to one that is already known. MR then becomes a possible

way of solving it. Thus, although it is already an important technique, MR is bound to be used to an even larger extent in the future.

In MR a preliminary model of the crystal structure is obtained by first orienting (rotating) and then positioning (translating) the model molecule in the crystal lattice. Having found the correct orientation and position we can calculate phases from the model and combine them with the observed structure factor amplitudes. The structure factors thus obtained, and the corresponding electron density map, contain a strong bias towards the starting model but they are usually sufficiently close to the correct values for a successful refinement.

More information on MR may be found in the Study Weekend proceedings (Machin, 1985; Dodson *et al.*, 1992).

7.3 The rotation function

The rotation function is used to find the correct orientation of the model in the crystal lattice. This may be achieved by testing the agreement between the Patterson functions calculated from the model and from the data at various relative orientations. The concept is simple: the rotation function can be thought of as a convolution in a rotational space. At the relative orientation that superposes the model onto the crystal, the product of the two Patterson functions—which are maps of interatomic vectors—should have a large value. The function was originally formulated as

$$R(C) = \int_U P_1(\mathbf{x})\, P_2(\mathbf{x}')\, \mathrm{d}V, \tag{7.1}$$

where P_1 and P_2 are the two Patterson functions, C is a matrix defining a rotation of $\mathbf{x}$ with respect to $\mathbf{x}'$, and U is the volume of integration. For mathematical convenience this can be reformulated as an integral over all space:

$$R(C) = \int_{\text{all space}} P_1(\mathbf{x})\, U(\mathbf{x})\, P_2(C\mathbf{x})\, \mathrm{d}V, \tag{7.2}$$

where $U(\mathbf{x})$ is defined equal to one inside the sphere of integration and zero elsewhere. In reciprocal space this becomes

$$R(C) = \sum_{\mathbf{h}} I_{\text{obs}}(\mathbf{h}) F_{M2}(C\mathbf{h})), \tag{7.3}$$

where the $I_{\text{obs}}(\mathbf{h})$ are the observed intensities and $F_{M2}(C\mathbf{h})$, the Fourier transform of $U(\mathbf{x})P_2(C\mathbf{x})$, is a continuous function defined over all reciprocal space. This formulation, first used by Lattman (1970), is conceptually clear and flexible in application but unfortunately slow to compute.

The usual way to compute the rotation function is by the fast Fourier method (Crowther, 1972). It uses the idea that, since we are rotating spherical volumes, it should be more appropriate to expand the Patterson density in a sphere in terms of Bessel spherical harmonics.

The following are some guidelines and practical points to consider when calculating and interpreting the rotation function.

7.3.1 Model

Firstly one should try to use the best search model available. Refined coordinates should be used whenever possible. The importance of this may not seem obvious especially as the data we use in the calculations do not usually extend beyond 3.0–3.5Å. In practice, however, the difference may be between a clear solution and no solution at all. In general

one should also remove from the model any atoms that are known to differ from the target structure. For example, side chains that are known from sequence alignment to be different should be trimmed down to alanine. Similarly, differences in the loop regions—the insertions in particular—should be removed.

On the other hand one should not remove too much of the model and should bear in mind that structure is generally conserved better than sequence and that two corresponding pieces of chain in related proteins may differ chemically but still have a similar fold. Including many wrong atoms in the model increases the overall noise level, whereas removing too many correct atoms reduces the signal we are looking for. Here one tries to balance having a complete model and having a wrong one. If several related structures are known one could try to construct the 'best' model by assembling from previously superposed models the pieces that are most homologous to our protein.

It is sensible to try several models if they are available. Interpretation is easier if they are all oriented in the same way to begin with.

7.3.2 Resolution limits

Next we consider how best to maximise the signal we want to extract from the calculation. First of all, what resolution limits are most useful? For a case of identical structures we would use the data to the highest available resolution. In practice the model is not identical with the expected structure and a resolution limit of at least two or three times the expected mean coordinate difference should be used. Low resolution terms should be omitted as they contain information mostly about solvent rather than protein. This means that the terms lower then 10 Å should be omitted.

7.3.3 Patterson integration radius

As we have previously said, the Patterson function is a collection of interatomic vectors; some will be vectors between atoms within the same molecule i.e., *self vectors*, and some will be vectors between neighbouring molecules i.e., *cross vectors*. In the rotation function we try to include the self vectors and to eliminate the cross vectors as much as possible.

We start by placing the model molecule in a large cell with no symmetry (space group P1). If the smallest intermolecular distance is larger than the largest dimension of the molecule, all the cross vectors are eliminated as the limit of the Patterson synthesis does not normally exceed the dimension of the molecule.

For data it is impossible to eliminate all the cross vectors as the molecules in a real crystal lattice are closely packed, but the optimal radius of summation can be estimated. This depends a lot on the shape of the molecule. For a spherical molecule the optimal radius of summation would be the diameter of the molecule, as it would contain all the self vectors. For a very elongated molecule the choice is difficult. The longest self vector will be equal to the length of the molecule but at that radius there will also be many cross vectors. In such a case one would have to test various shorter radii, but in general there seems to be little point in including less than (say) half the self vectors. On the other hand some cross vectors in the summation should not be fatal. David Blow points out that fortunately most proteins have an approximately spherical shape and a of search radius of 75–80% of the molecular diameter includes $\sim$90% of the self vectors. Ian Tickle recommends using 75% of the *minimum* diameter of the molecule (the shortest edge of the minimal box into which it fits). Jorge Navaza has suggested using a radius equal to the maximum distance of atoms from the molecule's centre of mass, limited by the smallest dimension of the minimal box and such that the volume of the integration sphere doesn't exceed the volume of the box.

7.3.4 Model unit cell size

The choice of dimensions of the P1 unit cell for the model is straightforward. It must be at least the size of the molecule's minimal box plus the integration radius in each direction. Navaza suggests the 'at least' should involve adding the resolution to this sum.

7.3.5 Omitting weak terms

The rotation function is approximately proportional to $|F|^4$ so that it will be dominated by the largest terms. The small terms can usually be omitted from the calculations if computing speed is a problem. Fortunately, these days and with this type of problem, computing time is not usually a serious limitation. In general, if there are no compelling reasons, data should not be omitted.

7.3.6 Normalising

Normalising involves dividing data into resolution shells of constant volume and scaling the data so that the mean intensity does not decrease with resolution. (This is like calculating the E values used in direct methods). The scaling is usually done by applying an exponential 'temperature' factor to the structure factors. The normalised data are what we would get if atoms were like geometric points rather than spheres of electron density. Normalising increases the contrast of the Patterson map and the peaks should be better resolved.

If we decide to leave weak terms out of the calculation it is important that we do it after normalising. Thus we ensure that the reciprocal space is sampled evenly and not dominated by a few large, low resolution terms. Normalising can introduce series termination ripples in the Patterson but this has not been found to be a serious problem. The procedure can be recommended generally for both rotation and translation searches. A Bayesian justification is given by Bricogne in (Dodson *et al.*, 1992).

N.B. Normalising is related to but distinct from *sharpening* where high resolution intensities are scaled up by applying an exponential B factor.

7.3.7 Origin Removal

Patterson maps always have a strong origin peak. It arises from the null vectors i.e., from an atom to itself. Normally the origin peaks just add a constant term to a rotation function. There is, however, one case where the origin peak can cause problems. If the integration radius is close to one of the cell dimensions then the integration in Patterson space could include an origin peak one lattice translation from the origin, introducing large and meaningless artifacts into the rotation function.

The best way to avoid the problem is to keep the radius of integration smaller than the smallest cell dimension minus twice the resolution limit (i.e., the diameter of the origin peak). Alternatively we can subtract the origin peak. This can be done by dividing the data into resolution shells and subtracting from each shell its average. In fact this can conveniently be combined with the normalisation, because by definition the average E^2 is 1, so all one has to do to get an origin-subtracted normalised Patterson is to use $E^2 - 1$ instead of just E^2.

7.3.8 Non-crystallographic symmetry

In MR it is usually helpful to have non-crystallographic symmetry. Firstly, knowing the local symmetry can help us pick the correct solution of the cross rotation function.

7.3.8.1 Native Patterson

In general the effect of a non-crystallographic rotation axis is to produce a pseudo-Harker plane through the origin and perpendicular to the local symmetry axis. The Harker plane

transforms in the reciprocal space to produce a more easily interpretable peak in the rotation function in the direction of the rotation axis. In some cases, however, it is instructive to examine the native Patterson separately. In particular, when the local and crystallographic axes have the same order (e.g., both are two-fold) and are parallel, the combination of the two symmetry elements results in pure translation.

The Harker plane should have an identifiable peak where the intermolecular vectors coincide, and the position of this peak indicates the position of the molecular axis relative to the crystal axis. If the two axes are not exactly parallel the vectors do not coincide and the peaks become smeared out.

7.3.8.2 Self rotation function

Often there is more than one molecule (subunit) in the asymmetric unit. A *self rotation function* i.e., a rotation function where the native Patterson is compared with itself, can be calculated in such cases to find the transformation that superposes the different molecules. It is often difficult to decide whether a given peak is significant. The self rotation function has a massive origin peak, corresponding to zero rotation, and all the other peaks are relatively small. It is helpful to calculate the mean and standard deviation for all the values after the origin peak has been removed. We can take as a rule of thumb that if a peak exceeds five standard deviations it is significant. If it is weaker it may still be significant but it must be treated with skepticism.

The crucial test for the correctness of a solution is that it must be consistent with the cross rotation function. In addition, if we are dealing with a symmetric molecule i.e., a dimer or tetramer, the solution should correspond to the appropriate point group symmetry. A stereographic projection of, e.g. a $\kappa = 180^\circ$ (in polar convention) section of the rotation function can be plotted. It should contain all the peaks corresponding to two-fold axes. If, for example, we have a tetramer in the asymmetric unit (point group 222) the 180° section should contain a set of three orthogonal peaks 90° apart.

7.3.8.3 Concluding remarks

In a typical case several calculations of the rotation function are needed at various resolution limits and integration radii to find the correct solution. The correct solution does not always correspond to the highest peak in the rotation function but it should be the most persistent feature as the different parameters are varied.

It is worth trying to solve the rotation problem as accurately as possible as all the methods for finding the translational parameters are quite sensitive to mis-orientation errors.

7.4 The translation search

Having found the orientation of the molecule(s) in the asymmetric unit we now have to determine the translational parameters needed to position the molecule correctly relative to the symmetry axes. We might expect this to be easier than solving the rotation problem since the methods we use for this task involve correlating the complete set of structure factors or Patterson functions. In some cases, however, it turns out to be the more difficult part of the problem. The most likely reason for this is the errors introduced by the inaccurate orientation of the model. To get the translation search working well it is important that the rotational parameters are optimised. It is worth running translation searches for slightly different rotations.

The two main approaches to solving the translation problem i.e., the *R* factor search and translation functions are described next.

7.4.1 *R* factor search

This is the conceptually simplest used method for determining the translational parameters. It involves testing the agreement between the data and the structure factors calculated from a model placed at various positions in the unit cell. It is apparent that this is a computationally large task. It would seem that the calculation should involve looping over all atoms, for each calculated structure factor, for each position in the unit cell. The problem however is not quite so bad and we shall now see that there are some simplifying features.

Starting with model coordinates of the molecule in the correct orientation but in an arbitrary position, we can calculate the structure factors in a P1 cell with the same dimensions as in the crystal as

$$F_{\rm c}(\mathbf{h}) = \sum_{i}^{n} f_i \, e^{2\pi \mathbf{h}.\mathbf{x}_i}, \tag{7.4}$$

where the $\mathbf{x}_i$ are the coordinates of the $i^{\rm th}$ atom, f_i is the atomic scattering factor and $\mathbf{h}$ is the reciprocal lattice vector. The position of the $i^{\rm th}$ atom with respect to the crystallographic origin will be $\mathbf{x}_i + \mathbf{p}$, where $\mathbf{p}$ is the translation vector we want to determine. For m molecules in the unit cell there will also be atoms at $R_j(\mathbf{x}_i + \mathbf{p}) + \mathbf{t}_j$ with $1 \leq m \leq j$, where R_j is the rotation matrix and $\mathbf{t}_j$ the translation vector for the appropriate symmetry operation. The structure factor equation now becomes:

$$\begin{aligned} F_{\rm c}(\mathbf{h}) &= \sum_{j}^{m}\sum_{i}^{n} f_j e^{2\pi i \mathbf{h}.\left[R_j(\mathbf{x}_i+\mathbf{p})+\mathbf{t}_j\right]} \\ &= \sum_{j}^{m} F_j(\mathbf{h}) \, e^{2\pi i \mathbf{h}.(R_j \mathbf{p})}, \end{aligned} \tag{7.5}$$

where $F(\mathbf{h}) = \sum_i^n f_j e^{2\pi i \mathbf{h}.(R_j \mathbf{x}_i + \mathbf{t}_j)}$. The $F_j(\mathbf{h})$s—which are independent of $\mathbf{p}$—need only be calculated once, and the R factor for each $\mathbf{p}$ can be calculated without having to loop over all atoms for each value of $\mathbf{h}$.

In general the R factor varies only slightly as the model is moved throughout the asymmetric unit, so a sudden drop of a few percent is likely to be significant. Difficulties may arise mainly from two factors:

- The R factor is quite sensitive to misorientation of the model; a misorientation of a few degrees may be disastrous. If no clear solution is obtained initially, it may be worthwhile, though tedious, to repeat the calculation with the orientation slightly altered.
- The second point to remember is that the R factor varies steeply with the position of the model and if the function is not sampled on a sufficiently fine grid the solution may easily be missed altogether. If possible the grid points should be spaced no less than 1 Å apart.

Despite all the tricks to speed things up, the R factor search is still slower that the Translation Function. For example in a P_6522 problem, the Translation Function took 4.3 minutes on a microVax 3, on a 1 Å grid[2]. The R factor search for the same problem but using a coarser grid (2 Å) took 34 hours CPU time! On a 1 Å grid it would have taken 275 hours! Clearly in such a case the method of choice should be the Translation Function.

7.4.2 Translation functions

The translation function is conceptually related to the rotation function. Here we investigate the correlation between the observed intensities and the cross-vectors between the

[2]The time was mainly spent in the FFT.

symmetry related molecules of the model as it is moved about the unit cell. When the model is positioned correctly the function should have peaks at values corresponding to the translation vectors between the symmetry related molecules. There are several forms of the translation function to choose from but the T_2 function is normally used. A brief description of the main options follows. A more detailed description of the various translation functions is given in (Tickle, 1985; Tickle, 1992).

7.4.2.1 The T function

The T function was first proposed by Crowther and Blow (1967). It is derived from the general form of the translation function

$$T(\mathbf{t}) = \sum_{\mathbf{h}} I_{\text{obs}}(\mathbf{h}) F_{\text{c}}^2(\mathbf{h},\mathbf{t}), \tag{7.6}$$

where $F_{\text{c}}^2(\mathbf{h})$ is the calculated intensity and $\mathbf{t}$ is a vector defining the position of the test molecule. The calculated intensity can be expressed in terms of the molecular transforms of the individual symmetry-related molecules.

Thus for a simple example of two molecules in the unit cell related by a two-fold axis we get

$$F_{\text{c}}^2(\mathbf{h}) = F_{\text{c}}(\mathbf{h}) F_{\text{c}}^*(\mathbf{h}) = (F_{M1} + F_{M2})(F_{M1}^* + F_{M2}^*) \tag{7.7}$$

where F_{M1} and F_{M2} are the molecular transforms of the two molecules sampled at point $\mathbf{h}$ in reciprocal space, and * denotes complex conjugation. For a model molecule shifted by $\mathbf{x}$ from an original position only the phase of its molecular transform changes and F_M becomes $F_M e^{2\pi i \mathbf{h}.\mathbf{t}}$. The calculated intensities can now be expressed as follows

$$\begin{aligned} F_{\text{c}}^2(\mathbf{h}) = \quad & \left(F_{M1}(\mathbf{h})\, e^{2\pi i \mathbf{h}.\mathbf{t}_i} + F_{M2}(\mathbf{h})\, e^{2\pi i \mathbf{h}.\mathbf{t}_2}\right) \times \\ & \left(F_{M1}^*(\mathbf{h})\, e^{-2\pi i \mathbf{h}.\mathbf{t}_i} + F_{M2}^*(\mathbf{h})\, e^{-2\pi i \mathbf{h}.\mathbf{t}_2}\right). \end{aligned} \tag{7.8}$$

We should keep in mind that the two molecular transforms and their putative positions $\mathbf{t}_1$ and $\mathbf{t}_2$ are related by symmetry. We can now substitute the above expression into the translation function:

$$T(\mathbf{t}) = \sum_{\mathbf{h}} I_{\text{obs}}(\mathbf{h}) \left[F_{M1}^2 + F_{M2}^2 + F_{M1} F_{M2}^*\, e^{2\pi i \mathbf{h}.\mathbf{t}} + F_{M1}^* F_{M2}\, e^{-2\pi i \mathbf{h}.\mathbf{t}}\right], \tag{7.9}$$

where $\mathbf{t}$ now equals $\mathbf{t}_1 - \mathbf{t}_2$.

We are only interested in terms determining the cross vectors i.e., dependent on the relative positions of molecules, so we can omit the first two F terms which represent the self vectors for the two molecules. For simplicity one of the cross terms is also omitted. The result is known as the T function.

$$T(\mathbf{t}) = \sum_{\mathbf{h}} I_{\text{obs}}(\mathbf{h}) F_{M1} F_{M2}^*\, e^{-2\pi i \mathbf{h}.\mathbf{t}}. \tag{7.10}$$

7.4.2.2 Removing the self-vectors: the $T1$ function

In an attempt to improve the signal-to-noise ratio we may try to subtract the coefficients for the self vectors from the the observed intensities

$$I_{\text{cross}} = I_{\text{obs}} - k\left(F_{M1}^2 - F_{M2}^2\right). \tag{7.11}$$

This rests on the approximation that the search model and the structure have the same self vector set. It should be pointed out that correct scaling is critical for the self-vector subtraction to work.

The complete formula for the $T1$ function as it is usually defined can now be written:

$$T1(\mathbf{t}) = \sum_{\mathbf{h}} I_{\text{cross}}(\mathbf{h}) F_{M1} F^*_{M2} e^{-2\pi i \mathbf{h}.\mathbf{t}}, \tag{7.12}$$

where

$I_{\text{cross}}(\mathbf{h})$ are the observed intensities with the the self vectors subtracted;
F_{M1} is the Fourier transform of one of the molecules in the crystal;
F^*_{M2} is the complex conjugate Fourier transform of a second molecule;
$\mathbf{h}$ is the reciprocal vector *hkl*;
$\mathbf{t}$ is the translation vector between molecule $M1$ and molecule $M2$.

The function $T1$, defined in this way, is an ordinary Fourier summation and can be evaluated by a fast Fourier algorithm.

7.4.2.3 $T2$ function

Whereas the $T1$ function has to be calculated for every pair of symmetry related molecules the $T2$ function combines all the symmetry. It has the form

$$T_2(\mathbf{t}) = \sum_{\mathbf{h}} I_{\text{cross}}(\mathbf{h}) \times 2 \sum_i \sum_{j<i} F_{Mi} F^*_{Mj} e^{-2\pi i \mathbf{h}.\mathbf{t}}. \tag{7.13}$$

The $T2$ function applies phase shifts so that all the peaks pile up at the same place, defined by a single translation vector $\mathbf{t}$ for the reference molecule, so the signal/noise is much better than for a single $T1$ function. This is implemented by `tffc`.

7.4.2.4 The TO/O function

This is an attempt to generalise the translation search by combining a 3-dimensional translation function with a packing function (Harada *et al.*, 1981). This is implemented by `tffc`. Some of the published results are impressive.

The function has the form

$$T_H(\mathbf{x}) = TO(\mathbf{x})/O(\mathbf{x}) \tag{7.14}$$

where $TO(\mathbf{x})$ is like the translation function T (§7.4.2.1) with a normalisation factor, and has the form

$$TO(\mathbf{x}) = \frac{\sum_{\mathbf{h}} I_{\text{obs}}(\mathbf{h}) I_{\text{c}}(\mathbf{x},\mathbf{h})}{\sum_{\mathbf{h}} I^2_{\text{obs}}(\mathbf{h})}. \tag{7.15}$$

$O(x)$ is a packing function that measures the interpenetration of the molecules:

$$O(\mathbf{x}) = \frac{\sum_{\mathbf{h}} I_{\text{c}}(\mathbf{h},\mathbf{x})}{N \sum_{\mathbf{h}} F_M(\mathbf{h})}. \tag{7.16}$$

Chapter 8. Phase Improvement by Density Modification: Solvent Flattening and Molecular Averaging etc.

You boil it in sawdust, you salt it in glue;
You condense it with locusts and tape,
Still keeping one principal object in view —
To preserve its symmetrical shape.
— The Hunting of the Snark

8.1 Density Modification

8.1.1 The Problem

Approximate electron density maps phased by such methods as MIR or anomalous scattering may or may not be enough to deduce the structure of the molecule. Examination shows that they fall into two groups:

1. Those maps in which there are visible structural features, and which it is possible at least to attempt an interpretation. Once a significant portion of the map has been interpreted it is often possible to use this information to improve the rest of the map.
2. Those maps which do not appear to contain structural motifs, from which it is not possible to attempt an interpretation.

The distinction may seen obvious, but in fact we could imagine a third case: when a map is not good enough to show the correct structure, then it could clearly show an incorrect structure. The fact that an experienced worker can distinguish an interpretable map from an uninterpretable map suggests that it may be possible in some manner to quantify the property of map quality. If we the modify the map in such a manner to increase this quantity then it may be possible to convert a uninterpretable map into an interpretable one.

8.1.2 The Method

How are good electron density maps distinguished from poor ones? Even without knowing the structure of a particular protein there are some features which can be expected to appear in the correct map, such as:

Flat solvent. The solvent in which the protein is crystallised will fill the gaps caused by imperfect packing of the molecules. This solvent is usually disordered from cell to cell, and so in the diffraction structure the electron density will be constant to a first approximation. This is applied to an estimated map through the process of solvent flattening (Wang, 1985; Leslie, 1988).

Predictable distribution of density values in the protein. Most proteins have fairly similar proportions of atomic types, distributed throughout the cell according to known constraints of atomic spacing. As a result the distribution of density values in the protein region is fairly similar from protein to protein. This information is applied through the process of histogram matching (Zhang & Main, 1990).

Histogram matching employs the predictable atomic makeup of biological macromolecules to predict the histogram of density values in the protein region. The current density map can then be systematically modified to bring it into consistency

with the predicted histogram. This technique is complementary to solvent flattening, since solvent flattening operates on the whole of the solvent region and histogram matching operates on the whole of the protein region. The known density histogram is a weaker constraint on the electron density than solvent flatness, but the volume involved is usually larger, and histogram matching has more power for phase extension than solvent flattening. The combination of solvent flattening and histogram matching usually converges in 10–20 cycles.

Connectivity of electron density. Proteins consist of linearly connected chains of peptide units. A good map will show this linear connectivity, and it is this feature which provides the starting point for building a model of the structure. The connectivity of the map can be enhanced by a process of iterative skeletonisation, as developed in the PRISM package (Baker *et al.*, 1994).

Known properties of the structure. Some features of the structure may be known from analysis of the X-ray data, or from biochemical sources. These include the presence of non-crystallographic symmetries relating different parts of the map, or relationships with other molecules, from which a portion of the structure can be deduced. This information can be applied through non-crystallographic symmetry averaging (Bricogne, 1974) and molecular replacement respectively.

Molecular averaging can be used in the case of non-crystallographic symmetry (NCS). NCS-related regions of the map are averaged together, thus reducing the signal-to-noise ratio in the density. The NCS-relationships and masks must be determined before this calculation can be performed.

Atomicity. In the high resolution limit, the electron density map for a structure will show resolved atomic features. Atomicity can be enhanced by modifying the map to satisfy Sayre's equation (Zhang & Main, 1990; Cowtan & Main, 1993).

Sayre's equation is a technique from small molecule direct methods, which applies the constraint of atomicity to the density. Protein structures do not normally diffract to atomic resolution; however, in the case of a good starting map and observed magnitudes at close to 2.0 Å Sayre's equation can be a powerful tool for phase improvement.

Note that all of these features can be distinguished in a map without knowing the desired structure. Techniques have been developed by which a poor density map can be altered in a sensible manner in order to bring it into line with each of these constraints, these are set out in more detail in the references given above.

All these techniques depend on knowledge of the molecular boundary. dm can calculate this automatically by Wang's method (Wang, 1985; Leslie, 1988) if the fraction of the unit cell occupied by solvent is known.

The phase improvement calculation is cyclic and involves modification of the phases in order to obtain best agreement with the known constraints on the electron density and the observed structure factor data. This process is shown diagrammatically in fig. 8.1.

8.2 Using dm

dm provides an automatic procedure for using the techniques described above. A single run will read an initial reflection file, apply multiple cycles of density modification and produce an updated reflection file.

The initial reflection file must contain at least the observed structure factor magnitudes and their deviations, and the estimated phases and their weights. In the case of an MIR dataset the phases and weights will come direct from the MIR phasing (typically from the program mlphare). It is also possible to use MAD data, although typically the phase

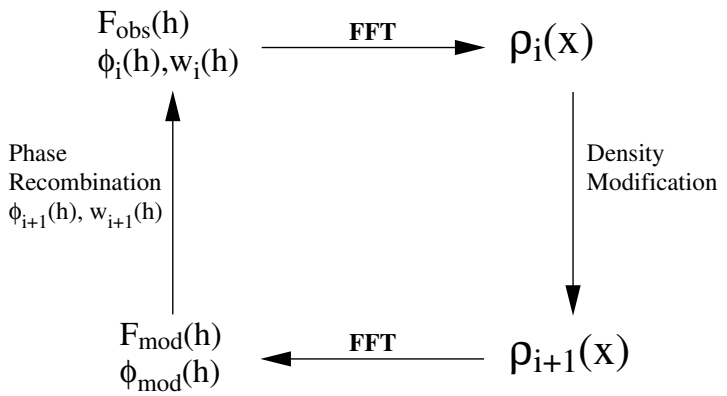


Figure 8.1: Phase improvement schematic.

extension scheme will be different since there is usually some phase information at all resolutions. This sort of calculation has proven particularly effective.

Density modification can also be applied in a molecular replacement calculation. In this case the correctly oriented model is used to generate a new set of structure factor magnitudes and phases. The agreement between the observed and model magnitudes is then used to generate weights for the model phases (typically using `sigmaa`'s `PARTIAL` option). The calculated phase and resultant weight are then input into the density modification calculation.

The use of sensible defaults means that the program can be operated very simply. In the case of a non-crystallographic symmetry averaging calculation, it is necessary to generate a mask covering one non-crystallographic subunit (usually one molecule), and in addition the non-crystallographic symmetry matrices must be included in the command file.

The most common application of the program involves use of the histogram matching and solvent flattening methods. These are very quick ($<$ 1 minute for 10 cycles with a unit cell volume of $\sim 10^5$ Å^3 on a typical workstation) and in the majority of cases lead to a good map improvement. In the case of very poor starting maps, the histogram matching can lead to a very broken map, in which case it may help to apply a temperature factor to smooth the map, or else repeat the calculation with solvent flattening alone. Averaging is always useful if the symmetry elements and mask can be determined. Sayre's equation is usually only helpful when the starting map is very good, and its application is very slow.

Other features available in `dm` include input of a user-defined solvent mask, output of the internally calculated solvent mask, averaging of multiple non-crystallographic symmetry related domains with different matrices, and automatic refinement of non-crystallographic symmetry matrices. The output of the program can be examined using `xloggraph`. The most useful indicator of whether the method has been successful is the 'free *R* factor'. This is analogous to the free *R* factor used in coordinate refinement in that it is calculated using a set of reflections which are excluded from the initial map calculation. However, once phase relationships between structure factors are introduced it becomes impossible to completely isolate a set of reflection since the absence of a set of reflections leads to a systematically error in the phasing of the rest of the reflections. Thus it is necessary to change the free *R* set from cycle to cycle, or if time is not a factor to run each cycle twice with and without a free *R* set.

The density modification free *R* is a property of both the density modification technique and the initial data set, so it can not be used to compare different density modification techniques. During a typical calculation the density modification free *R* will drop from $\sim$0.60 to $\sim$0.45.

In the case of averaging calculations, the correlation between regions related by non-crystallographic symmetry provides a good indication of whether the mask and matrices have been correctly determined. In later cycles this agreement will have increased through the averaging of the regions, and so becomes less informative.

The solvent flattening and molecular averaging techniques are closely related and are most effective when used together They are both noise filtering techniques and involve modification of electron density. They can be used to improve and extend phases.

8.3 Using Solomon

Several stages are required for phase refinement with Solomon. The stages described here assume that the phase probability distributions were determined experimentally (i.e. SIR, MIR, MAD).

- Determine phase probability distributions, described by Hendrickson- Lattman coefficients. This is done automatically by `mlphare`.
- Calculate the map to be flattened. Initially, you will have a map calculated from experimental phases. It is advised that you start from a resolution which has got significant phase information and do not yet extend the resolution!
  Subsequent maps can be calculated from coefficients produced by `sigmaa`. This gives the advantage that missing reflections can be estimated from the corresponding Fc (calcuated from the flattened map). The procedure can be improved by substituting in this way for the low resolution reflections that are missing.
- Have a look at the map and compare it with the original FOM weighted map. If there is no non-crystallographic symmetry, you might get some additional improvement by playing around with the solvent multiplication factor and the truncation level (see the keywords "TRUNC" and "SLVMUL"). If the crystals contain a lot of solvent (70 to 80%) you might try phase extension, but make sure the map actually improves by doing so. If you have got very weak phase information at higher resolution, try including it from the start, but keep the RADIUS at the lower resolution. (If you have got reasonable phase information to 3.7Å, and very weak information to 3.2Å, use a radius of 3.7Å, and a high resolution cutoff of 3.2Å; don't extend to 3.2, but use all information from the first cycle).
- Identify and refine non-crystallographic symmetry if present. The Uppsala programmes "O", "mama" and "imp" are ideally suited for this purpose. You must remove overlap between symmetry related masks. The mask can be either in "O" or "CCP4" format.
- The mask will encompasses a monomer. Each mask has a set of associated symmetry operators which descibe how the density within this mask is related to other density within the asymmetric unit. This is a bit different from the way things are done in the "rave" package and has some advantages.
- include the non-crystallographic restraints in "Solomon" and run the script again. You might need to reduce the level of truncation a bit and the solvent multiplier should probaly be a bit less negative.

8.4 Estimating solvent content

Estimating solvent content

This section summarises the results of (Matthews, 1968) for solvent content.

Matthews' number is:

$$V_{\rm M} = \frac{\text{cellvolume } (\text{Å}^3)}{M n_{\rm asymu} \text{nmols}_{\rm asu}} = \frac{V}{MZ} \tag{8.1}$$

where:
M is the protein molecular weight in Daltons
V is the unit cell volume
$n_{\rm asymu}$ is the number of asymmetric units
$\text{nmols}_{\rm asu}$ is the number of molecules in the asymmetric unit
Z, the number of molecules in unit cell $= n_{\rm asymu}\text{nmols}_{\rm asu}$.

Molecular weight is:

- something you get off a gel;
- *or* number of protein residues in molecule $\times 110$—very roughly!
- *or* number of non-hydrogen protein atoms in molecule $\times 14$—roughly!

(Use `rwcontents` to read your PDB file if you have one; it will count the number of atoms of each type. The output of `truncate` provides information on the cell contents.)

Matthews found $V_{\rm M}$ between 1.66 and 4.0, corresponding to protein contents of 30–75%, but proteins with higher solvent contents will give higher values of $V_{\rm M}$. (Solvent content of 90% $\Rightarrow V_{\rm M} = 12\ldots$.

Calculate $V_{\rm M}$ assuming $\text{nmols}_{\rm asu} = 1, 2, 3$ etc.... You *may* be able to narrow down the number of possibilities for $\text{nmols}_{\rm asu}$.

Turning this into the fraction of protein in the asymmetric unit:

$$V_{\rm p} = \frac{MZA_{\rm V}}{VN_{\rm A}D_{\rm p}} = \frac{A_{\rm V}}{N_{\rm A}D_{\rm p}V_{\rm M}} \tag{8.2}$$

where:
$V_{\rm p}$ is the fraction of protein volume in the asymmetric unit
$D_{\rm p}$ is the density of protein $= 1.35$ (Matthews, 1968)
$A_{\rm V}$ is the average atomic volume in Å^3, $= 10$ approximately
$N_{\rm A}$ is Avogadro's constant.
(This is the same as Matthews' formula: $V_{\rm p} = 1.66v/V_{\rm M}$; $1.66 = A_{\rm V}/N_{\rm A}$, $1/D_{\rm p}$ is Matthew's $v = 0.74$ cc/g.) Alternatively: $V_{\rm p} = N_{\rm p}A_{\rm V}/N$, where $N_{\rm p}$ is the number of protein atoms in the unit cell (including hydrogens). (There are about the same number of hydrogens as C N O etc.)

If $V_{\rm p}$ is the fraction of protein volume in the asymmetric unit, the density

$$\begin{aligned} \rho &= \rho_{\rm p}V_{\rm p} + \rho_{\rm s}(1 - V_{\rm p}) \\ &= 1.35V_{\rm p} + 1.0(1 - V_{\rm p}) \\ &= 0.35V_{\rm p} + 1.0 \end{aligned} \tag{8.3}$$

where $\rho_{\rm s}$ is the solvent density (1.0 for water). Therefore

$$V_{\rm p} = (\rho - 1.0)/0.35. \tag{8.4}$$

If you know the density you can work backwards and find the number of molecules in the asymmetric unit exactly. (`axissearch` will tell you the volume of the unit cell in Å^3.)

Now you get this useful information:

- Possible molecular weight per asymmetric unit;
- Fraction of protein/solvent (from 8.4);
- If you know the number of atoms in your protein subunit, (roughly 14 times the number of residues, including hydrogens) and saying roughly again that the volume per atom is 10 $Å^3$ it is possible to get an estimate of number of subunits/asymmetric unit. Even if you do not know the density accurately it is often possible to guess how many subunits there might be per asymmetric unit.

Often there is only one possible solution for N_{subunits}.

Chapter 9. Refinement and validation

I *don't believe there's an atom of meaning in it.*
— Alice's Adventures in Wonderland

Refinement has been covered in three Study Weekends, (Machin *et al.*, 1980; Goodfellow *et al.*, 1989; Dodson *et al.*, 1996); see also (Tronrud, 1994). The principal CCP4 refinement program is now `refmac` which is undergoing active development. Information in this chapter is liable to become out-of-date, and the reader is recommended to read the latest `refmac` documentation.

9.1 Least squares structure refinement

The CCP4 suite provides the program `restrain` for least squares structure refinement. The old program prolsq has now been made obsolete by the Maximum Likelihood program `refmac`, see section 9.2. The least-squares program `restrain` incorporates a number of differences from `refmac` (see section 9.1.1) and may be useful.

Alternative refinement programs include `TNT`, `CNS`, `SHELX93`.

9.1.1 Comparison of `restrain` and `prolsq`

Here is some discussion of the differences between the least squares refinement program `restrain` and the obsolete prolsq. Most of these points also apply to a comparison of `restrain` and `refmac`.

`restrain` does constrained anisotropic thermal parameter refinement using the TLS (translation/libration/screw-rotation) model. Unconstrained anisotropic refinement is not feasible without atomic resolution data (i.e. 1 Åor better), so this is out of the question for all but a handful of very small proteins. The end result of the TLS analysis should give some insight into secondary structure or domain motions. `refmac5` now does TLS refinement as well.

A major difference of approach is that prolsq uses an FFT for structure factor and derivative calculations, whereas `restrain` uses slow FT's. This means that prolsq takes many cycles ($\sim$50) to converge but each cycle is very fast, whereas `restrain` takes only a few cycles ($\sim$5), but each one is much slower. Normally prolsq has the advantage here.

`restrain`'s functionally is much like prolsq's with some small differences:

- prolsq treats all main-chain peptide residues as though they had identical geometry; recent data indicates that glycine and proline are different from the others. `restrain` treats them differently.
- `restrain` uses individual distance constraint weights based on the estimates of the standard deviations of (Engh & Huber, 1991) (these values are all in the dictionary together with the ideal distances). prolsq uses blanket values for the weights, because its dictionary doesn't contain the s.d.s.
- `restrain` has a coupled occupancy refinement option for disordered sidechains.
- `restrain` has a full-matrix option for estimating individual positional standard deviations. However it requires a *lot* of memory, and at present needs to be compiled with a re-parameterised include file.

- prolsq has the option of applying non-bonded intermolecular repulsion restraints (i.e. between symmetry-related molecules) as well as the intramolecular ones. At present `restrain` only applies the intramolecular repulsions.
- There is a subtle difference in the way the planar groups (peptide groups plus PHE, ARG etc. sidechains) are treated. prolsq restrains to the plane calculated from the coordinates before each refinement cycle, whereas `restrain` restrains to the current best plane; this should allow the planes more flexibility of movement.

9.2 Maximum Likelihood refinement

The maximum likelihood approach to model refinement has been implemented in the CCP4 program `refmac`. The `refmac` program is used for the restrained or unrestrained refinement or idealisation of a macromolecular structure. It minimises the model parameters to satisfy a Maximum Likelihood residual. There are options to use different minimization methods. `refmac` produces an MTZ output file containing `sigmaa`-style coefficients suitable for the calculation of $mF_o - DF_c$ and $2mF_o - DF_c$ maps using `fft`.

The latest version of REFMAC is significantly different to earlier versions and is known as "`refmac5`". The key functionalities of `refmac5` are:

- Restraints are calculated within the main program. A large dictionary of standard geometries is included. Restraints can also be created for novel ligands, see in particular the Monomer Library Sketcher in `ccp4i`,(which is an interface to the `libcheck` program).
- TLS refinement can be used. This is particularly useful when there is significant anisotropy, but the resolution does not warrant refinement of individual anisotropic displacement parameters (U values).
- A bulk solvent correction is calculated within the program, see the SOLVENT keyword.
- For atomic resolution data, full anisotropic refinement can be performed with anisotropic displacement parameters being refined for some or all atoms.
- If good experimental phases are available then they can be included in the maximum likelihood target. The accuracy of experimental phases, as described by the Figure of Merit or the Hendrickson-Lattman coefficients, is often overestimated, and a blurring function is provided to compensate for this.
- Rigid-body refinement can be performed, and may be useful in the early stages of refinement. One or more rigid-body domains can be defined via the RIGIDBODY keyword.

9.2.1 TLS refinement in `refmac5`

TLS refinement in `refmac5` results in:

- TLS parameters for each defined TLS group, held in the `TLSOUT` file and in the header of the `XYZOUT` file.
- Residual B factors output in the ATOM lines of the `XYZOUT` file. These B factors do not include any contribution from the TLS parameters.

The `XYZOUT` and `TLSOUT` files can be passed to the program `tlsanl`, which will analyse the TLS tensors and also derive individual anisotropic displacement parameters from the TLS parameters.

9.3 Automated model building

Victor Lamzin's Automated Refinement Procedure program (renamed `arp_warp` due to a clash with a Unix command) can be alternated with a refinement program such as `refmac5`

to automatically build or rebuild parts of a model. `arp_warp` updates the model by identifying and removing poorly defined atoms and adding new atoms. Rejection of atoms is carried out on the basis of the density interpolated at the atomic centre, the deviation of the density shape from sphericity and some distance criteria. Addition of atoms is performed on the basis of difference density coupled with distance constraints.

CCP4 distributes an older version of `arp_warp` (version 5.0) which has been renamed as `arp_waters`, and which should only be used for adding waters while cycling with `refmac5`.

9.4 Difference map generation

On completing a round of refinement, various types of difference map can be generated with the program `fft` for comparison with the current model (e.g. using the graphics program `O`). `refmac` produces weighted map coefficients suitable for $mF_o - DF_c$ and $2mF_o - DF_c$ maps: these coefficients reduce model bias and are recommended over the unweighted $F_o - F_c$ and $2F_o - F_c$ maps. To obtain weighted coefficients from the output of `restrain`, the program `sigmaa` can be used.

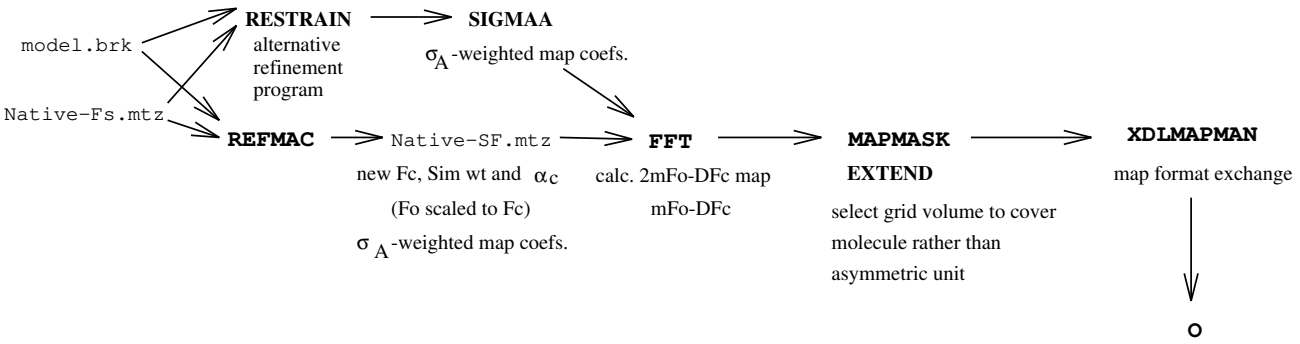


Figure 9.1: Steps in generating an electron density difference map.

9.5 Why is protein refinement difficult?

Small molecule people manage analysis and refinement with very few problems. Macromolecular crystals present several particular problems in refinement.

- For macromolecular crystals, the unit cell is big, and there are a very large number of X-ray data to collect, all of which have low signal-to-noise ratio. It is therefore not usually possible to collect data to atomic resolution as is normal for small molecule structures. The data available often suffer from both systematic and random errors. These are due to the crystal size, problems of mounting, absorption and crystal decay.
- Protein crystals have an additional problem. There is usually a high solvent content, and the crystal forces are weak. Some parts of the chain may not be crystalline at all, and others may have high thermal motion. This means that not all the unit cell can be properly parameterised. This is true for almost all proteins, not just those which diffract to lower resolution. This problem particularly reduces the intensity of the high resolution data. In addition it leads to severe effects of radiation damage.
- These two problems mean that experimental data extend to limited resolution, typically to a maximum limit in the range 3–2 Å.

- This means that the ratio of observations to parameters to be fitted is too low for conventional least-squares minimisation to converge.

9.6 Free R factor

A *free R factor* may be calculated by excluding a randomly-chosen fraction of reflexions from the refinement (Brünger, 1992)—a special case of the technique of cross-validation (A. T. Brünger, 1995). The agreement between their F_P and F_c is independent of the refinement procedure. `freerflag` may be used to add a column of tags to an MTZ file to label this set of reflexions. This is also included in the `uniqueify` script, which should be run on a dataset as soon as possible (see `unique` documentation). Note the CCP4 convention for this differs from `X-PLOR`—see the `freerflag` documentation. `f2mtz` and `mtz2various` may be used respectively to import and export `X-PLOR` and `SHELX` datasets with a free R flag to CCP4 taking into account the different conventions.

Since the deviation in R_{free} is roughly proportional to its value divided by the square root of the number of reflections, a test set of about 1000 reflections should be acceptable. Both R_{cryst} and R_{free} are global measures which cannot detect local errors. If atoms are placed in correct positions the R factor will decrease even if they are chemically inappropriate. NCS will reduce the value of R_{free} and different types of NCS will have different effects. Any pseudo lattice where the NCS does not increase the reciprocal space sampling cannot easily be utilised for refinement.

9.7 Validation, gross and overall errors

9.7.1 Validation

An lot of validation requires common sense. For example a good structure will not have most of its torsion angles in strange parts of the Ramachandran plot. Other obvious, but very useful, checks include:

- Are there unacceptable symmetry contacts between adjacent molecules?
- Are the B factors sensible, e.g. higher at the surface than in the core, not wildly divergent between adjacent atoms?
- Does the chemistry make sense? e.g.: do the H-bondable groups actually make H-bonds? Are there charged groups buried in hydrophobic environments?
- Do the maps show the expected features, e.g.: do omit maps reveal the missing atoms; do difference maps show substrate atoms?
- Is there a suspicious divergence from NCS?

Most programs flag many of the above.

9.7.2 Errors

Serious mistracing is rare but can happen. The existing tools (R_{free}, stereochemical checks as applied in `procheck` and `what-if`) easily detect such errors if applied sensibly. Actually the Ramachandran plot alone is a powerful tool for cross validation to identify such gross errors since the dihedral angles are not usually used as restraints in refinement programs.

Local errors such as loops out of register can easily be overlooked or ignored. They can be identified by the real space R factor and B values as a function of residue number.

Overall imprecision and refinement not taken to convergence is difficult to detect but happens—there are many examples of structures refined first against one data set to a certain resolution, then re-refined against a different higher resolution data set. The final structure gives a lower R factor against the original data than the one refined

against that data. This is may be due to poor weighting of prior and experimental information.

9.7.3 Bad practice

9.7.3.1 Not using all the available data

- Do not use a low resolution cut-off, e.g. many structures are still reported as being refined with data in the resolution range 5–2 Å. The data within the 5 Å shell contain a wealth of important information on your structure.
- Make sure you have not lost all the strong (often low resolution) terms through detector saturation—especially important with image plates at synchrotrons. Make a second data collection pass, or even a third, to avoid this. The *big* terms dominate all steps in your structure analysis.
- If you can possibly avoid it do not leave a large wedge of data uncollected. Offset your crystal by up to 15° to avoid a blind region. Make sure you cover the appropriate rotation range, and start at an appropriate orientation.

9.7.3.2 Attempting refinements when the observation to parameter ratio is too low

- At about 2.8 Å for a protein crystal with about 50% solvent, the number of observations is equal to the number of positional (*xyz*) atomic parameters. Even at this resolution the least-squares minimum is no longer well defined. Unless there is non crystallographic symmetry (or extremely high solvent content) it is therefore *foolish* to "refine" against data sets at resolutions below 2.8 Å. If there is NCS this limit can be relaxed with care, always ensuring the number of parameters is less than the observations: this absolutely requires the NCS to be imposed. Caveat: if your NCS is close to pseudo crystallographic symmetry (e.g. $P2_12_12$ but pseudo I222 or $P6_5$ pseudo $P6_522$), then it is less powerful and you will have special problems.
- Do not try to refine individual isotropic atomic B values till you have enough observations to about 2.5 Å or better.
- The significance of introducing extra parameters should be cross validated using $R_{\rm free}$.

Chapter 10. The Fast Fourier Transformation — or some of the things you always wanted to know about the FFT but were afraid to ask

"Of course," the Mock Turtle said: "advance twice, set to partners—"
"—change lobsters, and retire in same order," continued the Gryphon.
— Alice's Adventures in Wonderland

10.1 Introduction

There must be very few algorithms that have made as great an impact on science as the Fast Fourier Transform (FFT).[1] It was Lynn Ten Eyck (1973; 1977) who introduced the FFT to crystallography and we have been using it ever since. Whenever we do something that involves a calculation of an electron density map or structure factors i.e., moving between real and reciprocal spaces, it is almost certain that the program we use has an FFT algorithm embedded somewhere inside it. We can tell if it does not—the calculation is orders of magnitude slower than what we have come to expect. In addition to the above uses, the FFT has an important role in molecular replacement, in the calculation of the rotation and translation functions. This development is mostly due to Tony Crowther (1972), who showed how the rotation function can be calculated by an FFT if the Patterson function is expanded in spherical harmonics, and David Blow (Crowther & Blow, 1967), who applied the FFT to translation functions.

10.2 How it works[2]

Although it is not necessary to know anything about the mechanism of the FFT in order to use it, it might still be interesting—especially if the explanation is reasonably understandable. A good description of the FFT is given in (Press *et al.*, 1986). It goes like this:

If we have a discrete Fourier transform of length N we can split it into two transforms of length $N/2$ each. One of them is formed from the even-numbered terms and the other from the odd-numbered terms:

$$\begin{aligned} F_h &= \sum_{j=0}^{N-1} f_j e^{2\pi i jh/N} \\ &= \sum_{j=0}^{N/2-1} f_{2j} e^{2\pi ih(2j)/N} + \sum_{j=0}^{N/2-1} f_{2j+1} e^{2\pi ih(2j+1)/N} \\ &= \sum_{j=0}^{N/2-1} f_{2j} e^{2\pi ijh/(N/2)} + W_h \sum_{j=0}^{N/2-1} f_{2j+1} e^{2\pi ihj/(N/2)} \end{aligned}$$

[1] C. F. Gauß reputedly used the equivalent of real-valued FFTs in 1805, but no surprise there. There were independent discoveries of FFT algorithms in modern times preceding that of Cooley and Tukey, often referred to as the discoverers.

[2] It is 'interesting' to note that there are various software patents connected with the FFT and similar transforms which monopolies may restrict your freedom to use these algorithms; (this is of course impossible). Such software monopolies are a serious threat to free software and quasi-free software such as CCP4. Oppose software patents and interface copyrights!

$$= \quad F_h^{\text{even}} + W_h F_h^{\text{odd}}, \tag{10.1}$$

where $W_h = e^{2\pi i h/N}$.

Depending on N we can continue dividing the original transform. This is known as *factoring*. We then get a series of terms like $F_k^{\text{even,odd,even...}}$, in this case, for a Fourier transform of points that are successively even, odd, even etc. in the successive subdivisions of the data. In the extreme case, when N is some power of two, we can divide the data all the way down to transforms of length one. A transform of length one is just an identity operation. Now the transform consists of terms like

$$F_k^{\text{even,even,odd,even,odd,odd... even}} = f_n. \tag{10.2}$$

The problem now is with book-keeping—we need to know what n is. The answer is as follows. Let even = 0, odd = 1 and then read the resulting patterns backwards i.e., reverse the bits, and you get in binary the value of n. It works because the successive subdivision of data into even and odd are tests of successive least significant bits of n. If we now rearrange the data into the bit-reversed order, the transform can be constructed by combining adjacent pairs of points to get two-point transforms, then the adjacent pairs of pairs are combined to get four-point transforms and so on until the first and the second half of the data is combined into the final transform.

If you have a mathematical or functional programming bent and want a deep understanding of the algorithm, consult (Jones, 1989) for a *calculation* of it.

10.2.1 Example

Let's take a simple example of an 8-point transform. The coefficients of Fs of the successive divisions of data are:

		e (=even)				o (=odd)	
	ee		eo		oe		oo
eee	eeo	eoe	eoo	oee	oeo	ooe	ooo

In binary this becomes

$$000 \quad 001 \quad 010 \quad 011 \quad 100 \quad 101 \quad 110 \quad 111,$$

and with the bits reversed:

$$000 \quad 100 \quad 010 \quad 110 \quad 001 \quad 101 \quad 011 \quad 111,$$

which in decimal corresponds to elements numbered 0 4 2 6 1 5 3 7. This means that we first have to combine element 0 with 4 (adjacent pairs on the above list), 2 with 6, 1 with 5 and 3 with 7 to get two-point transforms. (N.b., each element is normally a complex number).

Next we combine the resulting elements 0 with 2, 1 with 3, 4 with 6 and 5 with 7 to get 4-point transforms. Finally we combine the resulting elements 0 with 1, 4 with 5, 2 with 3 and 6 with 7 to get the complete 8-point transform. Done.

10.2.2 Why is it fast?

Let's now see how fast the FFT is compared with the traditional, 'brute force' approach. First the data have to be rearranged into the bit reversed order. That takes very little time and requires no extra memory as it only involves swapping pairs of data. The transform itself is constructed in $\log_2 N$ sweeps through the data i.e., the number of times all the N data points can be factored—three in the above example. At each level data points are

combined pairwise. This means the number of operations in each sweep is proportional to N. Thus the whole algorithm is of order $N \log_2 N$. Such $O(N \log N)$ asymptotic complexity is typically associated with 'divide-and-conquer' algorithms like this. The transform can be parallelised and vectorised.

What about the non-FFT approach? From the expression

$$F_h = \sum_{j=0}^{N-1} f_j e^{2\pi i jh/N} \tag{10.3}$$

we see that for every F_h, numbering N, we cycle over all f_js, (also numbering N). Thus the number of operations is proportional to N^2.

What is the difference between N^2 and $N \log_2 N$? For $N = 1000$ the factor is 100. For $N = 1000000$ the factor is 50000—roughly the difference between one minute and one month. Moral: beware of algorithms with quadratic (or higher) complexity.

10.3 Crystallographic FFT

So far we have only discussed the case where the number of data points N is a power of two. What if N cannot be factored all the way to single points? There are other, highly optimised, algorithms tailored for handling transforms of various lengths. In fact, the crystallographic FFT does not require N to be a power of 2, but in general N has to be a 'nice' number. No prime factors greater than 19 are permitted. Additional requirements have to be satisfied for specific space groups.

In crystallographic applications the FFT is additionally optimised to take advantage of crystallographic symmetry. The application of the FFT in crystallography is explained in full by Ten Eyck (1973; 1977; 1985).

Spacegroup-specific transforms use symmetry to speed up the calculation but are more likely to have bugs, especially in uncommon spacegroups. Use of P1 is thus recommended if speed isn't a problem; if it is, test the higher-symmetry version against P1.

10.4 Programs

The CCP4 suite contains several programs making use of the FFT, e.g: `fft` (based on Ten Eyck's original) calculates Fouriers, difference Fouriers, Pattersons and difference Pattersons from reflection data; `sfall` calculates structure factors and X-ray gradients for refinement using inverse and forward FFTs; `almn` calculates rotation function overlap values using FFT techniques; etc. ...

`fftbig` is an in-core version of `fft` which runs in P1 only. It may be faster you have enough *real* memory available—otherwise it will cause page thrashing. This has replaced the original fft program.

Part III

The propagation, care and feeding of a CCP4 installation

Chapter 11. Installation

README file: *n. By convention, the top-level directory of a UNIX source distribution always contains a file named 'README'. When asked, hackers invariably relate the README convention to the famous scene in Lewis Carroll's 'Alice's Adventures In Wonderland' in which Alice confronts magic munchies labelled "Eat Me" and "Drink Me".*
— (Raymond, 1993)

11.1 First off

The distribution is supplied in source form. You will need Fortran, C and C++ compilers! (See §13 for hints about freely-available systems if necessary.). To build `phaser` python is required.

By default the 'unsupported' programs will be built as well as the main set. You may want to avoid building them to save space. Make sure you have plenty of spare disc space to build in.

You need ∼250 MB free for $CCP4_MASTER if everything in ccp4 is downloaded off the web site (ccp4/phaser). The space taken by the full set of binaries will vary considerably with system and compiler options, but is ∼350 MB on an Iris compiled with the default configuration; you can reduce this either by playing with the configuration options or only by compiling a subset of the programs. If ccp4mg and coot are also downloaded the space requirements will increase to ∼1GB. Some of the programs use considerable amounts of scratch space in execution—you will need 10s of MB free in the directory pointed at by $CCP4_SCR (see `ccp4.setup` (Unix)). In some cases you may find a need for ∼100 MB of scratch space, depending on the programs you use and their input. Some programs take up large amounts of memory which can cause problems. Personal memory useage quotas can be exceeded or for small systems, like PCs, hard limits!

11.2 Directory structure

The directories under the `ccp4` directory which holds the CCP4 suite are as follows:

`bin` is the default destination for installed program binaries
`ccp4i` graphical user interface for the suite
`doc` plain text documentation files (generated from the `html` files)
`etc` Unix shell scripts
`examples` example scripts/DCL procedures
- **`toxd`** data and model coordinates for a small protein (alpha-dendrotoxin from green mamba venom: "toxd") used in the examples
- **`rnase`** data and model coordinates for a second protein (ribonuclease from Streptomyces aureofaciens) used in the examples
- **`unix`**
  - **`non-runnable`** scripts which can't be run for want of data
  - **`runnable`** scripts which can be run with the data in `toxd`
- **`tutorial`** contains procedural scripts in five main areas, namely MR, MIR, refinement, MAD and density modification.

html library and program documentation in HTML form

include setup scripts and the `.def` files needed for program startup (§3.7)

lib The default destination for installation of the binary library files

- **ccif** source code for the CCIF library
- **data** machine-independent 'library' files such as the symmetry operators
- **src** source code for the CCP4 library
  - **mmdb** source code for the MMDB library
- **clipper** source code for pirate and the clipper libraries
- **ssm** source code for superpose and ssm libaries
- **fftw** source code for the MIT fftw libaries

man contains soft links to enable `man` command

manual LaTeX source for this manual and corresponding PostScript file

src program source

unsupported source for the 'unsupported' stuff

x-windows X-Windows programs (`xloggraph` etc.)

The `ccp4` directory itself contains top-level configuration files etc.

11.3 Building under Unix

11.3.1 Preliminaries

First decide where you want to put the sources for the software—this will be a directory we'll refer to as $CCP4_MASTER[1]—which will contain the `ccp4` directory. A conventional possibility for $CCP4_MASTER is `/public/xtal`, but somewhere in `/usr/local` is more in line with Unix conventions.

11.3.2 Unpacking the files

If you got the distribution by ftp or via the web site as gzip'd or compressed tar files, you can do the following:

`gunzip` ⟨*file*⟩ *or* `uncompress` ⟨*file*⟩
`tar xvf` ⟨*file*⟩

where ⟨*file*⟩ is replaced by the name of each `.tar.gz` or `.tar.Z` file in turn. This may be performed automatically by the supplied `install.sh`.

If you got the distribution on cd-rom, `cd` into $CCP4_MASTER and read the cd-rom using `tar`:

`tar xvf` ⟨*device*⟩

where ⟨*device*⟩ is the appropriate cd drive e.g., `/dev/cdrom` and the `v` option lists the file names as they come off (for comfort only). An additional option like `o` is necessary on some systems e.g., IRIX, to prevent the files' original ownership being retained.

If you don't want the 'aggregated' software—just CCP4—use

`tar xvf` ⟨*device*⟩ `ccp4`

11.3.3 Environment variables and `ccp4.setup`

The suite depends on some environment variables being set up appropriately. This is usually done by the script $CCP4/`include/ccp4.setup` (where $CCP4 is $CCP4_MASTER/`ccp4`). Copy the supplied `ccp4.setup-dist` in the `include` directory to `ccp4.setup` and

[1] Where the $ implies that this is the value of an environment variable.

edit it appropriately for your needs according to the instructions in the comments[2]. Note particularly the remarks in it about setting up for X-Windows programs; this is highly system-dependent.

With `ccp4.setup` edited to your satisfaction, execute it by

```
source $CCP4/include/ccp4.setup
```

to define the variables in the environment. Each user wishing to use CCP4 should have a similar line in their `.login` file, where $CCP4 is replaced by the appropriate *absolute* path (with any necessary qualification for shared filesystems).

11.3.4 Configuration

Now you're ready to *configure* the source directory for the peculiarities of your version of Unix. Here we assume that you are using the normal vendor-supplied compilers and that you've used the default paths suggested in `ccp4.setup` so that the suite is built and installed in the $CCP4 directory. If you have more complicated requirements, consult the full story on `configure` in §11.6. If you wish to do multiple installations for multiple machine types, then see below.

With $CCP4 as your current directory, type

`./configure` ⟨*system*⟩

where ⟨*system*⟩ describes one of the known operating systems corresponding to what you're running. This might be something like `irix`, `sunos`, etc. To find the possibilities, try

```
./configure help
```

`configure` will grind away and after some sanity checks e.g., on the values of the environment variables you've defined, will establish the correct directories and `Makefiles`. (If `configure` doesn't support your system you'll have to consult the porting guide (Chapter 15) and/or seek advice from CCP4.)

If you wish to do multiple installations for multiple machine types, then you should run the script `duptree` first. This will produce a further script containing information on the source directory tree. This second script should then be run once for each machine type — in each case a duplicate source tree is created with soft links to the original source. The `configure` script should then be run within each duplicate source tree. The script `duptree` contains further details.

11.3.5 Building

With a configured system you can now actually build the programs. Still in the $CCP4 directory, just type `make` and leave it to build. When you're satisfied with the build—see §11.3.6 for information on testing—you can use `make install` to copy things to the appointed places[3]. You can follow this by `make realclean` to tidy up by deleting the files you've copied from where they were built unless you want to do tests before overwriting an existing installation, for instance[4]. Alternatively, `make clean empty-targets` will

[2] `ccp4.setup` and the examples below mostly assume your shell is `csh` (or `tcsh`), where there is a distinction from `sh` and derivatives like `ksh` and `bash`, but if you don't use a `csh`-like shell you should have no trouble in adapting the instructions. There is, however, a `ccp4.setup-bash` version of `ccp4.setup` which `bash` users can use as a basis for a version to be sourced in `.bash_profile`

[3] You could use `make install` to start with if you're confident and there's no problem with clobbering a previous CCP4 installation.

[4] `make realclean` will delete the directories containing split files used to build the library and you will need these if you have to run a symbolic debugger with library code. In this case you may prefer to `make clean` and then `make empty-targets`; the latter will make empty files in place of the compiled programs, saving space, but preserving the time information that `make` can use later.

leave dummy copies of the binaries so that if the sources get updated, `make` will only rebuild the relevant ones[5]. The suite should now be functional and you can start doing some useful work with it. Although, the program documenation is in plain text and HTML form, when building the Suite `man` pages are also generated.

11.3.6 Testing

If you want to run tests before installing the suite, you have to put the compiled programs on your path; do (with `csh`):

```
set path=(⟨builddir⟩/src $path)
```

where ⟨*builddir*⟩ is the directory in which you ran `configure` ($CCP4 in the case described above). You'll find a small example dataset in $CCP4`/examples/toxd`, and a second one in $CCP4`/examples/rnase`, and some runnable scripts in $CCP4`/examples/unix/runnable`. Some log files generated at Daresbury are distributed for comparison.

11.3.7 Problems

Since the amount of testing of different systems we can do is limited, there may be problems with the installation on less widely-used systems, perhaps due to non-standard Fortran code which hasn't yet been eliminated, or peculiarities of different versions of Unix. First of all check our Web page (`problems.html`) which has a list of known problems with the current release. If there is no fix then please try to solve the problem locally, if you can and let CCP4 know about what changes you needed to make; if you can send us patches (made with `diff -c`) electronically, that would be most helpful. If you successfully use without changes a configuration which reports itself as 'untested' or 'not properly tested' it would be helpful if you could let CCP4 know that it works.

If you can't solve the problem, contact CCP4 (see §12.2) and we will help if we can (but no promises!). One thing you can do is to try building the library and running the test program for it with `make testlib` to look for clues—if you complain to CCP4 we will ask you for the results of this if we suspect a low-level library problem. In the meantime, if there are problems building some of the programs (`make` fails on the `srcdir` target, having done the `libdir` target successfully) you can build as much as possible using `make -i` and then install this much with `make instsome` in the `src` and/or `unsupported/src` directory. (The `instsome` target doesn't insist that everything is properly built first.)

11.3.8 Saving space and shared libraries

The program binaries are big, and you may need or want to make them smaller. There are various possible ways to do this which are system-dependent. Increasing the optimisation level of the compiler might make the binaries bigger or smaller. However, under agressive optimisation the programs may not work at all. Using shared libraries will definitely make them smaller. On some systems you can use the `configure` argument `--with-shared-lib` successfully to build a shared version of the CCP4 library to link against (but make sure you have the correct setup to find the shared CCP4 library at runtime, namely the location of the shared library should be included in the environment variable LD_LIBRARY_PATH). Other system supporting the `dlopen` mechanism, at least, should be able to use the library shared too. See §11.6 for ways to change the configuration parameters or use parameters to `make`, e.g.

```
make FOPTIM=-O
```

[5] If you can't keep the CCP4 directory around on disc, after `make empty-targets` you might back it up to cd with `tar` to preserve the date information if you subsequently get updates.

In extremis you can probably save some space by removing all debugging information from the binaries using the `strip` program or installing using `install -s`. This will prevent diagnosing some problems, though.

11.3.9 Installing updates

If you subsequently get updates as a complete distribution, copy it into the right places in $CCP4 as per §11.3.2 and use `make [install]` again; you probably don't need to re-edit `ccp4.setup`, but it's worth looking at the new version to make sure. Re-run `configure`, although this may not be strictly necessary. If bug-fix updates arrive as a *patch* file, you should copy this into $CCP4_MASTER and use the `patch` program to apply them[6]. A helpful tutorial on using `patch` is in the documentation for version 2.0 and later of the GNU `diff` implementation. See also http://www.ccp4.ac.uk/problems/patches.html

To build individual programs (the interesting ones you know have changed), run `make` ⟨*name*⟩ in $CCP4`/src` where ⟨*name*⟩ is the program name e.g., `fft`. When you have installed the changes to your satisfaction, a `make clean` will remove the `.orig` files left behind by `patch`. If you kept the directory intact after running `make empty-targets`, a simple `make` will just rebuild the necessary.

Before installing a new major version it is possible to remove the installed binaries from the previous one with `make uninstall`. In particular, this will remove obsolete binaries.

11.3.10 Installation summary

Here's a summary of the simplest procedure for installation (again, depending on the platform this may be performed automatically):

- `cd` $CCP4
- `gunzip` [if necessary]
- `tar x...`
- [edit `include/ccp4.setup`]
- `./configure` ⟨*system*⟩
- `make testlib` [not normally called-for]
- `make` [optional step]
- `set path=($CCP4/src $path)`, run some tests and re-set `path` [optional]
- `make install`
- `make clean` [or `realclean` or `empty-targets`] [optional, to tidy up]

11.4 CCP4I

From version 4.0, the suite includes a graphical user interface called ccp4i. The files for this reside in sub-directory $CCP4`/ccp4i`. Necessary environment variables are set in the main setup file `ccp4.setup`. ccp4i exists as a set of tcl scripts, and requires no compilation. Once the setup file has been sourced, it can be run simply by typing `ccp4i`.

ccp4i does however require a local installation of `Tcl/Tk` with the `blt` extension. Details on this and other installation issues can be found at http://www.ccp4.ac.uk/ccp4i_main.html

[6] You can get `patch` from the usual archives of free Unix software—it's distributed by the GNU project, for instance—or from the Daresbury info-server if necessary (see §13).

11.5 X-windows programs

If you wish to use the X-Windows-based programs such as `xloggraph` you will have to ensure that they have access to the *resources*' relevant *application defaults* files. Where these files are might depend on how you install the programs and the mechanism for picking up the resources is likely to vary from site to site. An example is given in `ccp4.setup` for Unix users. The X-Windows-based programs can now be built on SGi machines running irix, PC Linux-based machines and DEC machines, using the general CCP4 procedure - see the `--with-x` option below. If you are not using one of these systems consult the `README` files in the `x-windows` directory for instructions.

11.6 The full story on `configure`

§11.3 described a simple use of `configure`. This section explains its use in more complicated situations, where it really comes into its own[7].

The Unix configuration system allows you automatically and consistently to create the system-dependent `Makefiles` etc., to operate several systems with shared files in a distributed filesystem, and to test new versions without disrupting the production ones. It is based on a Bourne shell script, some `Makefile` skeletons and macro-processable program files which are manipulated under the script's control. The `configure` script encapsulates most of the information about the system dependencies in the suite and automates their application. The rest of this information resides in the macro-processor directives in `unix.m4` and the `.[ch]` files.

The main configuration parameters (apart from the system type) are controlled by command-line flags to `configure` of the form

$$--\langle flag\rangle{=}\langle value\rangle$$

where $\langle flag\rangle$ may be abbreviated uniquely and = may be replaced by whitespace. The currently-implemented flags of this form are are:

`--srcdir` The CCP4 source directory (referred to as $CCP4 in §11.3);

`--bindir` Where the program binaries get installed;

`--libdir` Where the library files get installed.

Other common options are controlled by flags of the form `--with-`$\langle feature\rangle$ or `--disable-`$\langle feature\rangle$:

`--with-f2c` Use the free `f2c` compiler rather than a 'native' one;

`--with-shared-lib` Build a shared version of `libccp4` and link against it.

`--with-x` Build the X-Windows-based programs on some platforms.

`--disable-`$\langle feature\rangle$ Disable build components. Among these options are `phaser`, `cctbx`, `clipper`, `pdb_extract`.

`configure` currently assumes that the shareable (plain-text) library files remain in $CCP4/`lib/inclu` rather than being installed elsewhere.

If the build is not done in $CCP4, `configure` creates `src` and `lib` directories mirroring those in $CCP4 and does the compilations there.

`configure` examines the environment variables necessary for running the suite as defined by `ccp4.setup` and tries to check them as far as reasonable, but these are not actually used in the build process. There are a number of configuration parameters that you don't usually want to change, but whose defaults can be over-ridden by the values of environment variables when `configure` is run:

CC the C compiler (default is system's native compiler; another possibility might be `gcc`);

[7] Those familiar with GNU software will recognise the influence behind this `configure`, but beware that it isn't quite like the GNU `configure` scripts.

COPTIM C compiler optimisation flags (default is the highest optimisation which is considered safe or unagressive);

XCFLAGS any extra flags you need to give to the C compiler, apart from those for optimisation (these are system dependent);

FC the Fortran compiler (default is system's native compiler); there isn't currently any support for other than native compilers;

FOPTIM Fortran compiler optimisation flags. The default varies, but will have no debugging extras. In some cases (notably IRIX), the default is used because there are so many problems encountered with the optimiser. You may want to try higher optimisation levels and see if there's a significant performance improvement and that the code still works...

XFFLAGS any extra flags necessary for the Fortran compiler, apart from those for optimisation; you might want to change this, for instance to add `-s` to make smaller binaries at the expense of debugging ability;

CXX the C++ compiler;

CXXOPTIM C++ compiler optimisation flags;

XCXXFLAGS any extra flags you need to give the C++ compiler;

XLDFLAGS any extra flags needed for `ld`, typically extra libraries;

F_LIBS the additional libraries included by the FORTRAN compiler;

C_LIBS the additional libraries included by the C compiler;

CXX_LIBS the additional libraries included by the C++ compiler;

RANLIB dummy on SysV Unix, `ranlib` on BSD, depending on whether `ar r` builds a symbol table itself;

LNS indicates how to make symbolic links in the filesystem (usually `ln -s`), else `cp` to copy rather than link. `ln` (hard links) could be used in the absence of symbolic links if you don't need to operate across file systems;

M4 how to run the `m4` macroprocessor and define a symbol to indicate the system type (see `configure` source);

MAKE the name of the 'make' program for use in recursive `Makefiles` if the system's make doesn't define the symbol `MAKE`. Usually null;

SETFLAGS set appropriately if individual programs need special Fortran compiler flags (most likely to suppress optimisation)—see the `configure` source;

INSTALL_PROGRAM a command to install executables. Uses `install -c` if a BSD version is available (to avoid interfering with running programs), else `cp`;

INSTALL_DATA a command to install non-executables. If BSD `install` is available, uses `install -m 644`, else `cp`.

There are corresponding `Makefile` variables that you can override, e.g.:

```
make FOPTIM=-g
```

Suppose you support several types of system and share the CCP4 source and system-independent data files across a network with a distributed file system such as NFS, AFS or RFS. Then it may be convenient to have a sub-directory in $CCP4 for each system in which to build and, possibly, in which to keep the resulting binaries. Directories for binaries can be anywhere, though.

As an example, a configuration used on the Daresbury Iris is given in fig. 11.1, which includes debugging options and scattered directories.

```
cd $CCP4
mkdir iris-build
cd iris-build
# debugging, diagnostics for C,
# some optimisation and debugging for fortran
env CC=gcc XCFLAGS='-g -Wall' FOPTIM=-g1 \
./configure irix --src=/ccpdisk/xtal/ccp4 --bin=/pxbin \
  --lib=~ccp4/lib
make
```

Figure 11.1: `configure` example.
(Don't follow this slavishly!)

Installing only the library

Some people require only the CCP4 library to support other systems and don't want to configure/install the CCP4 programs as such. Usually `configure` will expect to find the subdirectory `src` in the directory specified using the `--srcdir` flag, containing at least the makefile skeleton. You can avoid this requirement (so you don't have to fetch `src.tar.Z` by ftp, for instance) by using the flag `--onlylib` as an argument to `configure`. This will avoid configuring the `src` and `unsupported` directories. Build the library using `make onlylib` at the top level. The `--onlylib` flag may be abbreviated uniquely.

Chapter 12. Support, bug reports etc.

"Mine is a long and a sad tale!" said the Mouse, turning to Alice, and sighing.
— Alice's Adventures in Wonderland

12.1 Release policy

Continuous updates of the suite are no longer made. Instead there are intermittent releases of well-defined versions. Patches for serious bugs in the current version are made available by ftp (in the `prerelease` directory), and advertised on the `ccp4bb` mail list (see §13.3) and the Problems Page on the web (see §13.1). New major versions are also announced on `ccp4bb`, of course, and possibly by posts to appropriate network news groups. (Update notifications to non-commercial users are only made electronically and in the occasional CCP4 printed newsletter.) Releases may have version numbers of the form *n.m.p*, where *n.m* is the major version and *p* indicates the number of the 'patch' to that version, but at the time of writing this system has had to be dropped (this is a bug) and releases just have major version labels. A change of *n* in the version is intended to be used for substantial changes, e.g. when the file formats change in an incompatible way. The sources distributed by ftp and tape always correspond to the latest version, but 'patch files' of differences between different minor versions are available by ftp for the convenience of users wishing to upgrade from an earlier minor version.

12.2 Support and bug reports

Remember: *non-trivial programs generally contain bugs*. You may or may not feel that CCP4 has more than similar systems of the same size, but you do have the source to fix them if necessary.

Although CCP4 makes no commitment to support for the suite,[1] we are keen to improve it and will be grateful for reports of bugs (particularly with fixes!) or general constructive comments regarding functionality, usability etc., including installation and documentation. *The primary purpose of bug reports is to get problems fixed in a subsequent version for the good of all*, but we will often be able to furnish fixes or workarounds to avoid your research being stymied. Also, if you make significant enhancements to the software, please pass these on (as required by the licence). Programs not in the `unsupported` directory will get the most attention as they have someone specifically assigned to look after them.

Our ability to maintain different flavours of Unix is limited by their availability to us, but we are happy to include compatible changes supplied to us which are necessary for other platforms. Unfortunately, our VMS expertise and facilities are limited—most CCP4 development work is now done in Unix—but we do our best.... We cannot support operating systems other than VMS and Unix; if you need to run the suite under another OS, consult §15 in the first instance and we will be interested to hear of your experience.

Please make reports etc. to the CCP4 Secretary at Daresbury, preferably by electronic mail (to `ccp4@dl.ac.uk`). If you don't get a response within a few days, try again, but please make sure that it is possible to reply: problems we quite often encounter include

[1] Any mention of support by us in this document or elsewhere should be understood to be in quotes if not already so, to indicate that it isn't guaranteed.

invalid return paths (`From:` or `Reply-to:`) fields of your mail header with a bad network address and failure of the local system to deliver mail due to a bad user id or configuration problem. The `ccp4bb` list is not intended for bug reports.

We don't have time to answer queries not directly related to CCP4 software—please address questions about running system utilities etc. to your local computing support staff. Failing that, if you have access to it, Usenet news is the ultimate fount of wisdom (but not necessarily about CCP4 (!) and don't assume the developers read netnews). (See §13.6 and §13.4 for some information on obtaining non-CCP4 software.)

12.2.1 Reporting bugs

Before reporting a bug, please check whether it's been fixed in the latest version if you don't already have the latest. The distribution contains a `CHANGES` file which lists the major changes since the previous release. Next, check the Problems Page on the web site (see §13.1) to see if it has been fixed since the latest release. There is also a file called `PROBLEMS` in the distribution describing long-standing known bugs. In any case, it will usually be useful to know what version you are using (look in the 'banner' that the programs print on startup) and, if there might be some programming or compiler problem, what operating system you are using.

There is no orthodox standard for submitting effective bug reports, but some information that can typically be useful as well as mistakes to avoid is given in this advice adapted from Cygnus Support:

"In general, common sense (assuming such an animal exists) dictates the kind of information that would be most helpful in tracking down and resolving problems in software.

- Include anything *you* would want to know if you were looking at the report from the other end. There's no need to include every minute detail about your environment (users have been known to provide the values of every variable in their environments), although anything that might be different from someone else's environment should be included (your path, for instance). Often we will want to see the complete output ('log file') from the run.
- Narratives are often useful, given a certain degree of restraint. If a person responsible for a bug can see that A was executed, and then B and then C, knowing that sequence of events might trigger the finding of an intermediate step that was missing, or an extra step that might have changed the environment enough to cause a visible problem. Again, restraint is always in order ("I set the build running, went to get a cup of coffee (Columbian, cream but no sugar), talked to Sheila on the phone, and then THIS happened. . . ") but be sure to include anything relevant. [It is sometime helpful to know what (scientifically) you were hoping to do if this is unusual.]
- Richard Stallman writes, "The fundamental principle of reporting bugs usefully is this: *report all the facts*. If you are not sure whether to state a fact or leave it out, state it!" This holds true across all problem reporting systems, for computer software or social injustice or motorcycle maintenance. It is especially important in the software field due to the major differences seemingly insignificant changes can make (a changed variable, a missing semicolon).
- Submit only *one* problem with each problem report. If you have multiple problems, use multiple reports. This aids in tracking each problem and also in analysing the problems associated with a given program."

Chapter 13. Resources

Could you tell me, please, which way I ought to go from here?
— Alice's Adventures in Wonderland

13.1 CCP4 Web Pages

The CCP4 Web Pages are located at:

```
http://www.ccp4.ac.uk
```

We now make extensive use of these pages, and this should be your first stop when searching for information on CCP4 resources and activities.

13.2 Anonymous ftp

The CCP4 programs can be obtained from Daresbury via Internet anonymous ftp. It should be noted, however, that this in no way relieves the end user of the need to return a licence agreement. (A copy can be found on the CCP4 website at http://www.ccp4.ac.uk/ccp4license.html). Commercial users of CCP4 *must* obtain the right to access the source from CCP4 before any transfer takes place.
The current Internet address of this service is `ftp.ccp4.ac.uk` [IP number 148.79.112.134]. Users should log in with the id 'ftp' or 'anonymous' and their e-mail address as password. The sub-directory `ccp4` contains the relevant files. Transfers using this service are logged.

The files are held as 'compressed tar' files to reduce the size of them. There are several files described by a `README`.

A typical Unix ftp session may go as follows; not all output from ftp is shown, but the general idea is given with your input slanted. (Your `ftp` command may have a different interface.)

```
ftp ftp.ccp4.ac.uk
Connected to ftp.ccp4.ac.uk
Name : anonymous
Guest login ok, send your complete e-mail address as password.
Password: d.love@dl.ac.uk
          Welcome to the CCP4 ftp distribution service.
230 Guest login ok, access restrictions apply.
ftp> cd ccp4/current/linux
ftp> binary
ftp> get README —more
ftp> get ccp4-core.tar.gz
ftp> get phaser-cctbx.tar.gz
ftp> bye
```

You now need to uncompress and untar the files e.g.

```
cd $CCP4
tar xzf ccp4-core.tar.gz
```

13.3 Electronic mailing lists

13.3.1 The CCP4 Bulletin Board

The CCP4 Bulletin Board[1] is intended to be a forum for discussion, to query aspects of the CCP4 Program Suite, to publicise or announce meetings, etc., or to ask the user community for help. It is not intended for bug reports which should be sent to `ccp4@dl.ac.uk`. To send messages to and receive messages from the Bulletin Board, you need to subscribe (see below). The address to send messages to the Bulletin Board is `ccp4bb@dl.ac.uk`, and the system re-broadcasts these to the list of subscribers.

13.3.2 The Bulletin Board Subscriber Service

In order for you to use the Bulletin Board, you must first inform the system that you wish to subscribe to the list—subscriptions are handled by an automatic system. In its simplest form send the message `SUBSCRIBE CCP4BB` to: `Majordomo@dl.ac.uk`.

Note: take care with the address to manipulate your access to the list: a common error is to send the message to `ccp4bb@dl.ac.uk`, which results in the user community getting the message and no action taken by the system.

Messages to `Majordomo@dl.ac.uk` have blank lines ignored and all commands are case-insensitive. Your e-mail address will be worked out by the server, but can be changed with the `RETURN-PATH` command. To remove yourself from the list send `UNSUBSCRIBE CCP4BB` to `Majordomo@dl.ac.uk`. If necessary, just send `HELP` for guidance.

13.3.3 The developers' list

As well as the CCP4BB list, there is a list for people who consider themselves 'developers' of CCP4 code, on which policy is discussed, problems aired, fixes suggested, etc. To subscribe, send a message `SUBSCRIBE CCP4-DEV` to: `Majordomo@dl.ac.uk`. The list itself is `ccp4-dev@dl.ac.uk`.

13.3.4 Summary

Here is a summary of the relevant network addresses:

Majordomo@dl.ac.uk is for (un)subscribing to the CCP4BB and CCP4-DEV lists;
ccp4bb@dl.ac.uk is for broadcasting messages to the CCP4BB subscribers;
ccp4-dev@dl.ac.uk is for broadcasting messages to the ccp4-dev subscribers;
ccp4@dl.ac.uk is for contacting the Daresbury CCP4 staff for bug reports etc.;
ccp4a.dl.ac.uk is the address for retrieving files by anonymous ftp.

13.4 Other crystallographic software

Here is how to obtain some of the other crystallographic software mentioned herein but not distributed by CCP4. Network references are given as Uniform Resource Locators. A URL like `ftp://`⟨*site*⟩`/`⟨*directory*⟩ means anonymous ftp to ⟨*directory*⟩ at ⟨*site*⟩.

Babel `ftp://joplin.biosci.arizona.edu`
BIOMOL `ftp://rugcbc.chem.rug.nl/pub`, <cfb@chem.rug.nl>;
corels `ftp://sgjs1.weizmann.ac.il/pub/corels`;
demon `ftp://lccp.ibs.fr/dist/demon`; F.M.D. Vellieux <vellieux@lccp.ibs.fr>;
dssp Chris Sander <Sander@embl-heidelberg.de;
FRODO/TOM <frodo-request@biochem.ualberta.ca>
madnes Jim Pflugrath <pflugrath@cshl.org>;
madsys Bill Weis <weis@cuhhca.hhmi.columbia.edu>;

[1] 'Bulletin board' is a historical misnomer—this is just an electronic mailing list.

merlot Paula Fitzgerald, Merck Sharp and Dohme Research Laboratories, PO Box 2000, RY80M203, Rahway, New Jersey, USA 07065 <paula_fitzgerald@merck.com>;
molscript Per Kraulis, <per.kraulis@sto.pharmacia.se>;
mosflm ftp://ftp.mrc-lmb.cam.ac.uk/pub;
O, RAVE Alwyn Jones <alwyn@xray.bmc.uu.se>. See also http://kaktus.kemi.aau.dk/;
PDB information http://www.rcsb.org, http://www.ebi.ac.uk/msd;
PHASES W. Furey, VA Medical Center and University of Pittsburgh <300531@vm2.cis.pitt.edu>;
Photon Factory software (e.g. weis) ftp://pfweis.kek.jp;
Protein W. Steigemann <steigema@biochem.mpg.de>;
raster3D ftp://stanzi.bchem.washington.edu;
SHELX, PATSEE George Sheldrick, Institut für Anorganische Chemie, Universität Göttingen, Tammannstr. 4, D–37077 Göttingen, Germany; fax +49–551–393373 <gsheldr@shelx.uni-ac.gwdg.de>;
sftools ftp://mycroft.mmid.ualberta.ca;
Turbo Frodo http://afmb.cnrs-mrs.fr/TURBO_FRODO/turbo.html;
X-PLOR Axel Brünger <xplor@laplace.csb.yale.edu>;
Xtal Syd Hall <syd@crystal.uwa.edu.au>
XtalView <ccms-help@sdsc.edu>

13.5 Crystallographic information/discussion

'Bionet' provides a number of discussion/information forums by both e-mail and network news (equivalently). The 'bionet.xtallography' group discusses protein crystallography; there is also a network news group 'sci.techniques.xtallography'. The 'jobs' group might be of interest to some. For more information, e-mail biosci-help@net.bio.net or (preferably) see http://www.bio.net/.

The crystallography entry in the World Wide Web virtual library is at http://www.unige.ch/crystal/crystal_index.html.

WWW access to the BMCD crystallisation database is through http://ibm4.carb.nist.gov:4400/bmcd/bmcd.html.

13.6 Sources of general free software

Sources of free[2] software such as the GNU project distribution include the following anonymous ftp sites at the time of writing:

Asia ftp.cs.titech.ac.jp, utsun.s.u-tokyo.ac.jp:ftpsync/prep, cair.kaist.ac.kr:pub/gnu;
Australia archie.oz.au:gnu (archie.oz or archie.oz.au for ACSnet);
Europe src.doc.ic.ac.uk:gnu, ftp.informatik.tu-muenchen.de, ftp.informatik.rwth-aachen.de:pub/gnu, nic.funet.fi:pub/gnu, ugle.unit.no, isy.liu.se, ftp.stacken.kth.se, ftp.win.tue.nl, ftp.denet.dk, ftp.eunet.ch, nic.switch.ch:/mirror/gnu, archive.eu.net;
United States wuarchive.wustl.edu, ftp.cs.widener.edu, uxc.cso.uiuc.edu, col.hp.com, gatekeeper.dec.com:pub/GNU, ftp.uu.net:systems/gnu.

These are probably mainly, but not solely, appropriate for Unix stuff.

Even if you acquire the GNU software by ftp, please consider making a donation to the Free Software Foundation, e.g. by buying a tape or CD from them (CCP4 development depends on GNU software). Their address is:

[2] as in 'freedom', not necessarily price

Free Software Foundation
59 Temple Place - Suite 330
Boston, MA 02111-1307
USA
+1-617-542-5942 (voice), +1-617-542-2652 (fax) gnu@gnu.org

To search for a particular program in the sites registered in the system, get hold of the invaluable `archie` program. Usenet users might see also the FAQ posting for the Usenet news group comp.sources.wanted and postings to news.answers.

There is free literature on the network too. A case in point is `http://www.Germany.EU.net/books/carroll/alice.html`.

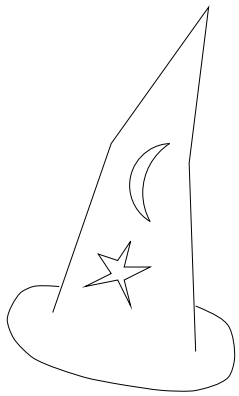
13.6.1 Free compilers

If your Unix system has an unbundled C compiler (i.e. one doesn't come with the system), you may be able to use GNU `gcc` and the GNU C libraries if you can get binaries or bootstrap them on another system. If you want support, there are several companies willing to sell it to you—see the distribution.

CCP4 will compile with (a suitably recent version of) the free `f2c` compiler, so you don't need a Fortran compiler either. `f2c`'s home is `ftp://netlib.att.com:netlib/f2c` but it's available elsewhere. The latest version of the GNU Fortran compiler `g77` will compile CCP4 with some effort, and remaining problems should be ironed out soon.

Part IV

Hackers' bit



Chapter 14. Writing and contributing programs

"It's my own invention"
— Through the Looking Glass

CCP4 is a *collaborative* project, and such collaboration is intended to encompass protein crystallographers anywhere. You're encouraged to write new software or adapt the CCP4 programs and donate the results to CCP4 for the benefit of the community. Below are some guidelines for doing so. The current software may not all obey these guidelines for one reason or another, but do as we say, not as we do!

If you're doing development work on the code you may want to join the `ccp4-dev` mailing list (§13.3.3).

14.1 General advice

Please, especially if you are writing new code, use the library routines where appropriate. Using library routines has various advantages from the point of view of maintainability, portability, consistency and (possibly) efficiency. *Abstraction is your friend.* If you see the need for new facilities in the library, please let CCP4 know. The library routines are (imperfectly) documented in the same place as the programs; if necessary use the source.

14.2 Requirements for CCP4 code

The following conditions should be satisfied by code to be included in the suite. (We are aware that not all the code in it currently fulfils these conditions, but . . .)

Language Most code is expected to be written in ANSI standard FORTRAN77 (ANSI, 1978), although we're prepared to accept some extensions which are in Fortran90 taken from (DOD, 1978), primarily long variable names, lower case, block `DO`, `INCLUDE` and `IMPLICIT NONE`. Don't use:

- `!`-style comments;
- `*`n qualifications on non-character declarations without good reason, like a serious need to save space—just `INTEGER`, `REAL`, `DOUBLE PRECISION`, `COMPLEX` or `LOGICAL` and no `BYTE`;
- system-specific subroutines (other than at the CCP4 library level);
- `CHARACTER` variables in `COMMON` with non-`CHARACTER`;
- non-standard ordering of declarations—`DATA` must come last.
- `FORMAT` statements without all items (including line termination descriptors) being separated by commas.

Besides ensuring portability, such restrictions allow us to use useful tools to check and maintain the code.

C is also acceptable for non-numerical applications but discouraged for numerical ones. Code in the library should not depend on ANSI C features since some people *still* have only 'K&R' C, but for non-library code they can be expected to acquire `gcc` if necessary. Try to make C POSIX.1-compliant (Zlotnik, 1991) and fix it up for those systems without the relevant features.

If you have to use Pascal it will need to be a dialect that can be grokked by `p2c` for translation to C.

Data formats Programs should use the standard data formats (§16) for reflexion, coordinate and map data.

Initialisation Fortran programs should call the routine CCPFYP initially, allowing one to run them using the mechanism

⟨*program*⟩ hklin ⟨*file*⟩ hklout ⟨*file*⟩ ...

Termination There should be *no* STOP statements within the code; rather, call the subroutine CCPERR. This has the form

CALL CCPERR(⟨*num*⟩, '⟨*string*⟩'),

where ⟨*num*⟩ is an exit value (0 for normal termination) and the ⟨*string*⟩ is a suitable message. Programs should terminate successfully with the call

```
CALL CCPERR (0, 'Normal termination')
```

Keywords wherever possible use the standard input keywords. Subroutines for handling the input of the most common keywords are provided in the parser subroutine library (or, probably better, the higher-level keyparse library).

Logical names Where possible the logical names for file connexion should have standard values, as follows (see also environ.def):

HKLIN Input reflexion file;
HKLOUT Output reflexion file;
XYZIN Input coordinate file;
XYZOUT Output coordinate file;
MAPIN Input map file;
MAPOUT Output map file.

The DATA and PRINTER files for parameter input and log output, respectively, are assumed to be preconnected, although it is good practice to call CCPDPN for them anyhow. If several files of one type are required then their file names should be derived by adding numbers to the names given above, e.g. HKLIN1, HKLIN2.

File opening There should be *no* OPEN statements in Fortran code. Instead call CCPDPN for general files, and appropriate library routines for the standard data formats. This avoids problems with such things as VMS READONLY.

Version printing A call to CCPRCS will be inserted to output version information if it is not present already.

MTZ history Programs which operate on MTZ files should use the subroutine LWHIST to write a line of informational text to the history header, stating at least the program name and the date on which it was run.

Documentation This should preferably be in HTML format. Plain text versions are produced at Daresbury using the lynx browser, for viewing with editors or as Unix 'man pages'.

Output It is helpful to instrument the output for xloggraph (see its documentation).

14.3 Non-portable features

There are a number of things to be wary of if your Fortran code is to be portable and thus acceptable to CCP4, in particular:

Uninitialised variables Don't assume you can use variables that haven't been assigned to or initialised in DATA;

```
      PROGRAM ⟨name⟩
      ⟨declarations⟩
C advertise the version:
      CALL CCPRCS(LUNSTO(1), '⟨name⟩', '$Date: 2006/02/05 16:53:43 $')
C call the pre-processor:
      CALL CCPFYP
C initialise the MTZ routines:
      CALL MTZINI
C open input MTZ file:
      CALL LROPEN(1, 'HKLIN', ...)
C open input map file:
      CALL MRDHDR(IN2, 'MAPIN', ...)
      ⋮
C close the files
      CALL LRCLOS(1)
      CALL MRCLOS(IN2)
      CALL CCPERR(0, 'Normal termination')
      END
```

Figure 14.1: Skeleton example CCP4 program using MTZ and map files.

Static allocation assumptions Don't assume that you can avoid `SAVE` where it is required by the Fortran standard, e.g. for local variables maintained between subroutine calls, `DATA` which are altered, `COMMON` not declared in `MAIN`, varying-length `COMMON` blocks;

Assumptions about storage layout Don't assume any more about storage than the standard dictates; you may not assume anything about the internal representation of values of different types or detailed argument-passing conventions; any bit-twiddling *must* be done with library routines;

Assumptions about arithmetic accuracy You can't assume either IEEE standard arithmetic, or VAX-type arithmetic—use double precision if necessary.

Otherwise, you need to observe the items indexed under 'restriction' in the standard (ANSI, 1978) such as on 'association of entities'. The `ftnchek` program is useful if your compiler is unhelpful about warning of non-standard/non-portable things (see 13.4). If `ftnchek` objects seriously to your code, we'll want it fixed. The (free) Toolpack software is even more useful (or the commercial NAGware Tools development of it).

14.4 Stealing code

There are many advantages to 'stealing' code from reputable sources, but make sure it is freely-distributable and you acknowledge it as required. In particular, note that the copyright conditions in Numerical Recipes(Press *et al.*, 1986) prevent code from there being used directly, and anyway, it is not well thought of by the experts. Free, solid numerical code can be had in abundance from the 'Netlib' archives by Internet ftp or mail-server.

14.5 Using the pre-processor in programs

To use the pre-processor in existing code, the only thing you need to add to your code is a call to the routine `CCPFYP` as the first executable statement in your program. This routine performs all of the pre-processing. It should be called before any other CCP4 library routines.

14.6 Library modules

Here is a brief list of the modules in the CCP4 library. Documentation on them is available, mostly as `.doc` files in the `doc` directory.

ccplib contains various utility routines which are potentially machine-dependent. It is built on VMS-specific code in `vms.for` and `vmsdiskio.for` and Unix-specific code in `unix.m4` and `library.c` (derived from the literate program file `library.nw`);

diskio contains routines for random access to stream-mode files, but most of the relevant code is actually in `library.c`. The VMS version is in `vmsdiskio.for`;

fftlib crystallographic FFT routines;

keyparse a higher-level interface to the `parser` routines;

maplib for handling CCP4-style map data;

modlib fairly random collection of mainly numerical analysis routines;

mtzlib reflexion file handling;

parser processing free-format input containing 'keywords'—it actually lexes more than parses;

plot84lib low-level graphics with PLOT84 metafiles;

plotsubs higher-level interface to PLOT84 (or, possibly, other) routines;

rwbrook for handling coordinate (PDB/Brookhaven) files;

symlib useful routines for handling symmetry operations.

Chapter 15. Porting CCP4

CCP4 is currently maintained for various flavours of Unix and for windows. It should be straightforward to port it to anything else with a (reasonably tolerant) FORTRAN77 compiler, C (preferably ANSI C) and POSIX.1 bindings for C (subject to the problems noted in §15.2). A POSIX.2 environment will probably allow you to use the Unix scripts. There should be nothing fundamentally preventing it running under other operating systems, but there are undoubtedly would be some practical problems doing this.

15.1 Portability features

There are several abstractions used to promote portability and implemented in the library (`ccplib`, `diskio` and the routines they call). These include

- Command line specification of file connexion by associating 'logical names' with file names;
- Calls hiding file opening specifics rather than using Fortran `OPEN`—one of the least portable Fortran features;
- Binary, seekable 'stream files';
- 'Bit-twiddling' routines for packing/unpacking, for instance, bytes to/from words;
- Access to some system services such as timing.

15.2 Fundamental operating system dependencies

Should you wish to undertake a port to a completely different operating system, to find operating system dependencies (which *should* be confined to the basic library modules), the first thing to do would be to look for occurrences of the `VAXVMS` call in the Fortran code. This will mainly concern filing system things. Some possibilities for assumptions which are made include:

- Hierarchical filing system with file names of the form $\langle foo \rangle.\langle bar \rangle$ and possibly rather long;
- ASCII character set;
- 32-bit words, 8-bit bytes, *not* middle-endian.

It is definitely assumed that the system has environment variables ('symbols' or 'logical names' in VMS DCL) that can be read and set from programs—although the library could implement its own dictionary mechanism instead—and that programs can interrogate their command line.

15.3 Unix-flavour dependencies

Most of the flavour- and compiler-specific stuff should be confined to the low-level routines defined in `library.c`[1], `binsortint.c`, `unix.m4` and the configuration variables set in the `configure` script. For a new port the first thing to do try is to run `configure` with argument `generic` to generate `Makefiles` etc. with all the defaults and see what you need to change. One thing you *will* need to find out is what your compiler's conventions are for calling C from Fortran and make the appropriate edits to `library.c` and

[1] This is not meant to be readable. It's derived from the *literate program*—check out `comp.prog.literate`—in the file `library.nw`, which can be used directly if you have the `noweb` tools mentioned in the code. It's done that way to be more maintainable at Daresbury.

`binsortint.c`. If the number representation of your architecture isn't picked up correctly by the `library.c` code, you may need to alter that. If you're lucky, this will be the same as one of the existing conventions and you won't have to do much editing. You may be able to guess other configuration parameters from existing ports or by trying first with `configure generic`. Edit a new the target into `configure` and try to run it and use the configuration to `make test` at the top level. When you've got that working, all *should* be OK; try it out, e.g. on the data in the `examples/toxd` directory.

It's worthwhile contacting the CCP4 staff before starting any porting work, for your benefit and ours.

Chapter 16. File formats

The information below about the format of reflexion, map and coordinate files is not normally required since libraries are provided for handling files in these formats. Information is stored in the headers of MTZ and map files to allow transparent reading on architectures other than that on which a file was written (see §3.6). For details of the 'na4' ASCII exchange format for `.mtz` and `.map` files, see the documentation and code for the program `mtztona4`. The obsolete LCF format for reflexion files is only relevant for converting old datasets—see the `lcflib` code.

The file formats are binary. We don't subscribe to the school of thought which advocates text-only files in this case of large amounts of floating point data, although they are a good idea in other cases. There are two basic reasons:

- Clear text files are substantially larger and consequently take more time to read from disc even if the decoding is relatively cheap and space isn't a consideration. An efficient text encoding such as NA4 presumably defeats the object or clarity;
- With clear text you can't (in general) preserve bit patterns on reading and writing and may lose accuracy, particularly on repeated i/o—consider 100 iterations. You're anyway probably at the mercy of the compiler's i/o library which may well not use the optimal algorithms for text↔binary conversion anyhow.

16.1 Reflexions (MTZ)

The MTZ[1] reflexion file format uses fixed length records with, in general, four bytes for each data item (REAL*4), with a minimum of 3 columns and currently a maximum of 200 columns of data per record—though these limits are imposed only by the software parameters. Additional information (title, cell dimensions, column labels, symmetry information, resolution range, history information and, if necessary, batch titles and orientation data) is contained in labelled header records. The columns of the reflexion data records are identified by alphanumeric labels held as part of the file header information.[2] The user relates the item names used by the program to the required data items, as identified by the labels, by means of assignment statements in the program control data.

The file contains basically two classes of records—header records and reflexion data records. A standard reflexion data file contains the following items, in the order given. Not necessarily all items have to be present.

- MTZ identification record, containing the 'machine stamp' encoding the number formats of the architecture it was written on as a full word at byte 8 from the start of the file;
- Reflexion data: columns of data held as REAL*4;
- A beginning of header record;
- A set of keyworded records containing;

[1] MTZ's Terribly Zany, but Means Tilde Zero—Mc Claughlin, Terry and Zelinka were its progenitors.

[2] This may bring to mind 'tables' or 'relations' in relational databases—intentionally so. The model for an MTZ file is two relations, one (the header) keyed on keywords such as `SYMMETRY`, the other (comprising the reflexions) keyed on the `H`, `K` and `L` attributes/columns. This formalism has been largely forgotten, but clarifies some things, particularly operations in `mtzutils` (or a replacement) which should implement some of relational calculus.

VERS Version stamp (Character*10, currently `MTZ:V1.1`)
TITLE File Title—short identification of file (Character*70)
NCOL number of columns, number of reflexions in file, number of batches (Integer). If the number of batches > 0 this indicates a multi-record file.
CELL Cell Parameters (Real(6))
SORT Sort order of 1st 5 columns in file (Integer(5))
SYMINF Number of Symmetry operations (Integer); Number of Primitive operations (Integer); Lattice Type (Character*1); Space Group Number (Integer); Space Group Name (Character*10); Point Group Name (Character*10).
SYMM Symmetry operations in International Tables style
RESO Minimum (smallest number) and maximum (largest number) resolution stored as $1/d^2$ (Real(2))
VALM Value with which Missing Number Flag is represented.
COL Column Label (Character*30); Column Type (Character*1) for each column; Minimum and Maximum value in each column (Real); ID of corresponding dataset (Integer)
NDIF Number of datasets represented in the file.
PROJECT ID of dataset (Integer); Project Name (Character*64). Normally one for each structure determination.
DATASET ID of dataset (Integer); Dataset Name (Character*64). May be several for each structure determination, e.g. for different crystals or derivatives.
DCELL Cell dimensions for a particular dataset. (Real(6))
DWAVEL Wavelength for a particular dataset (useful e.g. for MAD). (Real)
BATCH Batch Serial Number for each batch present (Integer). This line is not present if number of batches is 0.

- END of main header card;
- Up to 30 Character*80 lines containing history information;
- For multi-record files: Batch title (Character*70) and (optionally) orientation data for each batch present in the file;
- End of all headers record.

N.b. Column Types are an extra check that the user input assignment for a requested program label is of the correct type.

Normally the Miller indices will be in the first three columns, though in the definition of the format there is no restriction on the use of the columns of the reflexion data records. However, the subroutines which output the MTZ header information in a formatted way presume that the first 3 columns of a standard MTZ file are the Miller Indices, and the first 5 columns of a multi-record MTZ file are H, K, L, M/ISYM and Batch number.

16.1.1 Orientation data

The orientation block contains values of the orientation parameters, constraint flags for cell parameters, and the relative scale and temperature factor to put this batch on the same scale as the reference intensity:

1. Six cell dimensions (Ångström and degrees, real numbers). These define an orthogonalization matrix B to convert indices hkl to Cartesian crystal axes xyz.

$$\begin{aligned} x &= Bh \\ B &= \begin{pmatrix} a^* & b^*\cos\gamma^* & c^*\cos\beta^* \\ 0 & b^*\sin\gamma^* & -c^*\sin\beta^*\cos\alpha \\ 0 & 0 & \lambda/c \end{pmatrix} \end{aligned}$$

This gives x parallel to a^*, and y in the a^*b^* plane.

2. Six cell dimension flags (integer) defining which parameters may be varied.
   -1 parameter is freely variable;
   0 parameter is fixed (i.e. cell angle $= 90^\circ$ or 120°);
   $i > 0$ parameter is constrained to have the same value as parameter number i.
3. Orthonormal matrix U (9 real numbers in order $U_{11}, U_{12}, U_{13}, U_{21}$ etc.) to rotate the crystal Cartesian coordinates to the camera Cartesian coordinates. The `idxref` matrix is $A = UB$. The matrix U defines a standard orientation, and is a function of the mounting axis and the axis defined as $\phi = 0$, but it is not dependent on the misorientation angles. Typically, U will be a permutation matrix.
4. the 'mounting' axis, as defined for `oscgen` (1,2,3 for h,k,l)
5. Six misorientation angles, rotation axis mis-setting (six real numbers): $\phi_{x_1}, \phi_{y_1}, \phi_{z_1}, \phi_{x_2}, \phi_{y_2}, \phi_{z_2}$. The first three refer to the orientation at the beginning of the rotation range (at ϕ_{start}), the second three ϕ angles (if present) refer to to the end of the rotation range (at ϕ_{end}). The second set should be set equal to the first set if no crystal slippage is assumed.
6. Start and stop rotation angles, and range of ϕ values (3 real numbers) ϕ_{start}, ϕ_{end}, ϕ_{range}.
7. Crystal mosaicity in degrees.
8. $\lambda, \delta\lambda/\lambda$, correlated dispersion.
9. Scale and temperature factors for this batch (from `scala` etc.). The correction to be applied is
$$\text{scale} \exp(-2B(\sin\theta/\lambda)^2). \tag{16.1}$$

16.1.2 Standard column names and types

Standard names are normally used for the items as in fig. 16.1. The column types are as follows:

H	index h,k,l
J	intensity
F	structure amplitude F
D	anomalous difference
Q	standard deviation of anything: J,F,D or other
G	structure amplitude associated with one member of an hkl -h-k-l pair, F(+) or F(-)
L	standard deviation of a column of type G
K	intensity associated with one member of an hkl -h-k-l pair, I(+) or I(-)
M	standard deviation of a column of type K
P	phase angle in degrees
W	weight (of some sort)
A	phase probability coefficients (Hendrickson/Lattman)
B	BATCH number
Y	M/ISYM, packed partial/reject flag and symmetry number
I	any other integer
R	any other real

16.1.3 Missing Data Treatment

In a typical series of diffraction experiments, not all Bragg reflexions for a given resolution range are in fact recorded. Hence, after `truncate` some reflexion data records may be entirely missing from the MTZ file, although the reflexion indices lie within the measured resolution range. It is strongly recommended that index sets are made complete within the desired resolution range — a script to do this is provided in $CETC/`uniqueify`. The MTZ file will then contain records where there are indices but no measured data e.g.:

Name	Type	Item
H, K, L	H	Miller indices.
IC	I	Centric flag, 0 for centric, 1 for acentric.
M/ISYM	Y	Partiality flag and symmetry number[a]
BATCH	B	Batch number.
I	J	Intensity I.
I'	J	Selected mean intensity.
SIGI	Q	σI (standard deviation).
SIGI'	Q	$\sigma I'$.
FRACTIONCALC	R	Calculated partial fraction of spot.
IMEAN	J	Mean intensity.
SIGIMEAN	Q	σI_{mean}.
FP	F	Native F value.
FC	F	Calculated F.
FPHn	F	F value for derivative n.
DP	D	Anomalous difference for native data ($F^+ - F^-$).
DPHn	D	Anomalous difference for derivative n.
SIGFP	Q	σFP (standard deviation).
SIGDP	Q	σDP.
SIGFPHn	Q	σFn.
SIGDPHn	Q	σDPHn.
PHIC	P	Calculated phase.
PHIM	P	Most probable phase.
PHIB	P	Phase.
FOM	W	figure of merit.
WT	W	weight
HLA, HLB, HLC, HLD	A	*ABCD* H/L coefficients
FREE	W	free *R* flag (as program label)
FreeR_flag	W	free *R* flag (as file label)

[a] $256M + I_{\mathrm{sym}}$ where M is the partiality flag (0 for full, 1 for partial) and the 'symmetry number' I_{sym} is normally 0 (with $F = (F^+ - F^-)/2$ for anomalous data with both components measured) but is 1 if $F = F^+$ or 2 for $F = F^-$.

Figure 16.1: MTZ standard column labels and types (16.1.2).

0	0	2	MD	MD
0	0	4	517.0	23.0
0	0	6	1567.0	57.0
	...	...		

This means that it is easy to estimate completeness and later programs can "restore" values if possible. Furthermore, a particular reflexion may be recorded for the native protein but not for a derivative, and the corresponding combined reflexion data record should indicate "missing data" for the derivative.

To-date, missing data has been indicated in a variety of ways. For example, a zero standard deviation is taken to mean that the corresponding datum (e.g. structure factor amplitude) is missing. In all cases, however, the indicator is a number upon which arithmetic operations can (erroneously) be performed. This convention has now been discarded in favour of representing missing data as special values, namely IEEE NaN or VMS Rop. All relevant programs check for the presence of NaNs or Rops in input MTZ files, and take appropriate action. In particular, when displaying MTZ files using the program `mtzdump` (or the script $CETC/`mtzdmp`) missing data can be identified and are subsequently represented in the output in an unambiguous manner. Currently, missing data are given the value $1.E30$ which should be representable on all boxes, but which will overflow the format such that the output field is filled with asterisks.

All programs will now output NaNs or Rops where appropriate. Where such values occur in an input MTZ file, they will be carried through to the output. Alternatively, NaNs or Rops may be generated when for some reason no value can be calculated for a particular reflection and column.

A new program `mtzmnf` has been provided to convert old-style MTZ files to the new convention. This program relies on being able to identify "missing data", and to this end the following cases are checked:

(a) If SIGF= 0.0 then SIGF and the corresponding F (and D/SIGD if present) are replaced by NaN or Rop.
(b) If SIGI= 0.0 then SIGI and the corresponding I are replaced by NaN or Rop.
(c) If SIGD= 0.0 *and the reflection is acentric* then SIGD and the corresponding D are replaced by NaN or Rop.
(d) If the weight WT= 0.0 then PHIB, WT and the corresponding Hendrickson-Lattman coefficients A, B, C, D (if present) are replaced by NaN or Rop.
(e) If the calculated structure factor FC= 0.0 then FC and PHIC are replaced by NaN or Rop.

As a safety feature, only columns which are explicitly specified in the input to `mtzmnf` are converted. Old-style MTZ files may still be used with all CCP4 programs, and old-style checks on missing data remain in place (occurring after the check for a NaN or Rop). However, new data sets, completed with $CETC/`uniqueify` and combined with `cad`, should automatically include the necessary NaN and Rop entries.

16.2 Maps

The electron density map format was devised by Phil Evans. It can also be used for envelopes and images. Maps are stored in a randomly-accessible binary file as a 3-dimensional array preceded by a header which contains all the necessary information about the map. The header is organised as 56 words followed by space for ten 80 character text labels, as in fig. 16.2. Maps are structured as a number of *sections* each containing a (fixed) number of rows and each row contains a (fixed) number of columns.

1	NC	number of columns (fastest changing in map)
2	NR	number of rows
3	NS	number of sections (slowest changing in map)
4	MODE	Data type: **0** envelope stored as signed (8-bit) bytes (in the range -128 – 127); the convention is that values $\neq 0$ are 'true' (within the mask) although masks should normally only comprise values of 0 and 1. **1** image stored as (16-bit) half-words **2** image stored as (32-bit) reals **3** transform stored as complex 16-bit integers **4** transform stored as complex 32-bit reals Note: Mode 2 is the normal one used in CCP4 programs for maps and mode 0 is used for 'masks'; others haven't been tested recently and may not work.
5	NCstart	Number of first column in map
6	NRstart	Number of first row in map
7	NSstart	Number of first section in map
8	Nx	Number of intervals along x
9	Ny	Number of intervals along y
10	Nz	Number of intervals along Z
11–13	x, y, z	Cell dimensions (Å)
14–16	α, β, γ	Cell angles (degrees)
17	MAPC	Which axis corresponds to cols. (1, 2, 3 for X, Y, Z)
18	MAPR	Which axis corresponds to rows (1, 2, 3 for X, Y, Z)
19	MAPS	Which axis corresponds to sects. (1, 2, 3 for X, Y, Z)
20	Amin	Minimum density value
21	Amax	Maximum density value
22	Amean	Mean density value (Average)
23	ISPG	Space group number
24	Nsymbt	Number of bytes used for storing symmetry operators
25	LSKFLG	Flag for skew transformation, =0 none, =1 if foll
26–34	SKWMAT	Skew matrix S (in order S_{11}, S_{12}, S_{13}, S_{21} etc.) if LSKFLG$\neq 0$.
35–37	SKWTRN	Skew translation t if LSKFLG$\neq 0$. Skew transformation is from standard orthogonal coordinate frame (as used for atoms) to orthogonal map frame, as $X_0(\text{map}) = S(X_0(\text{atoms}) - t)$
38–52	future use	Some of these are used by the MSUBSX routines in `mapbrick`, `mapcont` and `frodo`. (All set to zero by default.)
53	MAP	Character string 'MAP ' to identify file type
54	MACHST	machine type stamp, as for MTZ files.
55	ARMS	Rms deviation of map from mean density
56	NLABL	Number of labels being used
57-256	LABEL(20,10)	10 80 character text labels (i.e. A4 format)

Symmetry records—if any—follow, stored as text as in International Tables, operators separated by `*x` and grouped into 'lines' of 80 characters (i.e., symmetry operators do not cross the ends of the 80-character 'lines' and the 'lines' do not terminate in a `*`).
Binary data follow as a byte-stream.

Figure 16.2: Map file format

Field #	Column (range)	Fortran format	Description
1	1–6	A6	Record ID (e.g., `atom`, `hetatm`)
2	7–11	I5	Atom serial number (residues in order beginning with amino terminus)
–	12	1X	Blank
3a	13–14	A2	Chemical symbol, right justified, e.g., '␣N'
3b	15	A1	Remoteness indicator e.g., `D`
3c	16	A1	Branch descriptor e.g., `1`
4	17	A1	Alternative location indicator (if any)
5	18–20	A3	Standard 3-letter amino acid code for residue
–	21	1X	Blank
6	22	A1	Chain identifier
7	23–26	I4	Residue sequence number
8	27	A1	Code for insertion of residue (if any)
–	28–30	3X	Blank
9	31–38	F8.3	Atom's x-coordinate (Å)
10	39–46	F8.3	Atom's y-coordinate (Å)
11	47–54	F8.3	Atom's z-coordinate (Å)
12	55–60	F6.2	Occupancy value for atom
13	61–66	F6.2	B (temperature factor)
–	67	1X	Blank
14	68–70	I3	Footnote number

Figure 16.3: PDB file coordinate record format.

16.3 Coordinates

The standard format adopted for coordinate data is that used in the Brookhaven Protein Data Bank[3]. The programs of the suite will handle either complete files or files containing only a subset of the types of record which may be present in a complete file. In particular the records containing the coordinate data (`ATOM` or `HETATM` records) are of interest. These PDB file records have the structure shown in fig 16.3. The Protein Data Bank provides a full description of the complete format (see §13.4).

A typical Fortran format for reading such records (as indicated by the third column of fig. 16.3) is

```
(A6,I5,1X,A4,A1,A3,1X,A1,I4,A1,3X,3F8.3,2F6.2,1X,I3).
```

The standard setting of the orthogonal axes for the Brookhaven format is:

$$x \parallel \mathbf{a} \qquad y \parallel \mathbf{c}^* \times \mathbf{a} \qquad z \parallel \mathbf{c}^*.$$

The suite assumes these settings if the `SCALE` cards are not present in a coordinate file.

16.4 PLOT84 graphics files

CCP4 currently uses a homegrown graphics metafile format called PLOT84, implemented by the routines in `plot84lib`. Its format might be relevant if you wanted to write a driver for a different graphics device, in which case see existing programs like `pltdev` as an example or consult the documentation in `doc/plot84.doc`.

[3]This format will eventually be replaced by the new macromolecular CIF format.

16.5 Library data files

Various data files used by the library routines and specific programs live in the $CLIBD directory:

`font84.ascii` data for creating the binary PLOT84 font file in $CCP4_LIB with `fontpack`—see the PLOT84 code and documentation;

`atomsf.lib`, `atomsf_neutron.lib` X-ray and neutron formfactors for every known atom (for `sfall`).

See documentation/code for the programs mentioned for further information.

16.5.1 Symmetry operators: `syminfo.lib`

Contains symmetry operators for the 230 spacegroups listed in (and following the conventions of) (Hahn, 1992) ('IT') and non-standard settings used by various CCP4 programs. Each space group has a header line comprising space-separated values of:

- Spacegroup number;
- Number of lines of symmetry equivalents ('positions' in IT) (n);
- Number of lines of primitive equivalents (p);
- Spacegroup 'short' name; subscripts are typed as-is and a prefix '`-`' represents an overbar, e.g. `P21/m` $\equiv P2_1/m$, `P-1` $\equiv P\bar{1}$;
- Point group name; the IT name is prefixed by `PG`; contrary to the spacegroup name convention, an overbar is represented by a trailing `bar`, e.g. `PG4bar3m`;
- Crystal system;
- Possible comments about non-standard settings etc.

Following the header are n lines of symmetry equivalents, of which the first p are the primitive ones.

Layout is such that:

- The symmetry operator lines are limited to 80 characters;
- The elements of operator triplets are separated by commas, and triplets are separated by `*` or newline.; the translations may be before or after the coordinate, e.g. `1/2+X` or `X+1/2`;
- The header lines should start in the first column and the other lines be indented (for ease of locating the headers).

Part V

Appendices

References

A. T. Brünger. 1995. The free *R* Value: A More Objective Statistic for Crystallography. *Methods in Enzym.*

ANSI. 1978. *American National Standard Programming Language FORTRAN*. American National Standards Institute, Inc.

Baker, D., Bystroff, C., Fletterick, R. J., & Agard, D. A. 1994. *Acta Cryst.*, **D49**.

Blow, D. M. 1958. *Proc. Roy. Soc.*, **A247**, 302.

Blow, D. M., & Crick, F. H. C. 1959. *Acta Cryst.*, **12**, 794–802.

Blow, D. M., & Rossmann, M. G. 1961. The Single Isomorphous Replacement Method. *Acta Cryst.*, **14**, 1195–1202.

Blundell, T. L., & Johnson, L. N. 1976. *Protein Crystallography*. London: Academic Press.

Bricogne, G. 1974. Geometric Sources of Redundancy in Intensity Data and Their Use for Phase Determination. *Acta Cryst.*, **A30**, 395–405.

Bricogne, G. 1991. A maximum-likelihood theory of heavy-atom parameter refinement in the isomorphous replacement method. *In:* (Wolf *et al.*, 1991).

Brünger, A. T. 1992. The Free *R* value: a Novel Statistical Quantity for Assessing the Accuracy of Crystal Structures. *Nature*, **355**, 472–474.

Castellano, E. E., Oliva, G., & Navaza, J. Fast Rigid-Body Refinement for Molecular-Replacement Techniques.

Collaborative Computational Project, Number 4. 1994. The CCP4 Suite: Programs for Protein Crystallography. *Acta Cryst.*, **D50**, 760–763.

Cowtan, K. D., & Main, P. 1993. Improvement of Macromolecular Electron-Density Maps by the Simultaneous Application of Real and Reciprocal Space Constraints. *Acta Cryst.*, **D49**, 148–157.

Crick, F. H. C., & Magdoff, B. S. 1956. The Theory of the Method of Isomorphous Replacement for Protein Crystals. I. *Acta Cryst.*, **9**, 901.

Crowther, R. A. 1972. The Fast Rotation Function. *Pages 173–178 of:* Rossmann, M. G. (ed), *The Molecular Replacement Method*. Int. Sci. Rev. Ser., no. 13. New York: Gordon and Breach.

Crowther, R. A., & Blow, D. M. 1967. A Method of Positioning a Known Molecule in an Unknown Crystal Structure. *Acta Cryst.*, **23**, 544–548.

Cullis, A. F., H., H. Muirhead, Perutz, M. F., Rossmann, M. G., & North, A. C. T. 1961. *Proc. Roy. Soc. A*, **265**, 15.

Dickerson, R. E., Kendrew, J. C., & Strandberg, B. E. 1961. The Crystal Structure of Myoglobin: Phase Determination to a Resolution of 2 Å by the Method of Isomorphous Replacement. *Acta Cryst.*, **14**, 1188.

DOD. 1978. *Military Standard, Fortran, DOD Supplement to American Standard X3.9–1978, MIL–STD–1793*. US Government Printing Office.

Dodson, E. J. 1976. A Comparison of Different Heavy Atom Refinement Procedures. *Pages 259–268 of:* Ahmed, F. R. (ed), *Crystallographic Computing Techniques*. Copenhagen: Munksgaard.

Dodson, E. J., Evans, P. R., & French, S. 1975. *Pages 423–436 of:* Ramaseshan, S., & Abramson, S. C. (eds), *Anomalous Scattering*. Copenhagen: Munksgaard.

Dodson, E. J., Gover, S., & Wolf, W. (eds). 1992. *Molecular Replacement*. CCP4 Daresbury Study Weekend, nos. DL/SCI/R33, ISSN 0144–5677. Warrington WA4 4AD, UK: Daresbury Laboratory, for Daresbury Laboratory.

Dodson, E. J., Moore, M., & Bailey, S. (eds). 1996. *Macromolecular Refinement*. CCP4 Daresbury Study Weekend. Warrington WA4 4AD, UK: Daresbury Laboratory, for Daresbury Laboratory.

Drenth, Jan. 1994. *Principles of Protein X-ray Crystallography*. Springer–Verlag.

Driessen, H., Haneef, M. I. J., Harris, G. W., Howlin, B., Khan, G., & Moss, D. S. 1989. *J. Appl. Cryst.*, **22**, 510–516.

Driessen, Huub P. C., & Tickle, Ian J. 1994. The use of normalised amplitudes in the Rotation Function. *Joint CCP4 and ESF–EACBM Newsletter on Protein Crystallography*, June.

Engh, R. A., & Huber, R. 1991. Accurate bond and angle parameters for X-ray protein structure refinement. *Acta Cryst.*, **A47**, 392–400.

Fan, Hai-Fu, Hao, Quan, & Woolfson, M. M. 1990. On the Application of One-Wavelength Anomalous Scattering. II An Analytical Approach for Phase Determination. *Acta Cryst.*, **A46**, 659–664.

Fitzgerald, P. M. D. 1988. MERLOT, n Integrated Package of Computer Programs for the Determination of Crystal Structures by Molecular Replacement. *J. Appl. Cryst.*, **21**, 273–281.

French, G. S., & Wilson, K. S. 1978. *Acta Cryst.*, **A34**, 517.

Germain, G., Main, P., & Woolfson, M. M. 1970. *Acta Cryst.*, **B26**, 274–285.

Giacovazzo, C., Monaco, H.L., & Viterbo, B. 1992. *Fundamentals of Crystallography*. International Union of Crystallography Texts on Crystallocraphy. Oxford.

Glusker, J. P., & Trueblood, K.N. 1985. *Crystal structure analysis: a primer*. Second edn. Oxford University Press.

Goodfellow, J., Henrick, K., & Hubbard, R. (eds). 1989. *Molecular Simulation and Protein Crystallography*. CCP4 Daresbury Study Weekend, nos. DL/SCI/R27, ISSN 0144–5677. Warrington WA4 4AD, UK: Daresbury Laboratory, for Daresbury Laboratory.

Green, D. W., Ingram, V. M., & Perutz, M. F. 1954. *Proc. Roy. Soc. A*, **225**, 287.

Hahn, Theo (ed). 1992. *International Tables for Crystallography*. third edn. Vol. A. Kluwer.

Hao, Quan, & Woolfson, M. M. 1989. Application of the Ps-function Method to Macromolecular Structure Determination. *Acta Cryst.*, **A45**, 794–797.

Harada, Y., Lifchitz, A., Berthou, J., & Jolles, P. 1981. A Translation Function Combining Packing and Diffraction Information: An Application to Lysozyme (High-Temperature Form). *Acta Cryst.*, **A37**, 398–406.

Harker, D. 1956. *Acta Cryst.*, **9**, 1.

Hayward, S., & Berendsen, H. J. C. 1998. Systematic Analysis of Domain Motions in Proteins from Conformational Change; New Results on Citrate Synthase and T4 Lysozyme. *Proteins, Structure, Function and Genetics*, **30**, 144.

Helliwell, J. R., Machin, P. A., & Papiz, M. Z. (eds). 1987. *Computational Aspects of Protein Crystal Data Analysis*. CCP4 Daresbury Study Weekend, nos. DL/SCI/R25, ISSN 0144–5677. Warrington WA4 4AD, UK: Daresbury Laboratory, for Daresbury Laboratory.

Hendrickson, W. A. 1991. Determination of Macromolecular Structures from Anomalous Diffraction of Synchrotron Radiation. *Science*, **254**, 51–58.

Hendrickson, W. A., & Lattman, E. E. 1970. Representation of Phase Probability Distributions for Simplified Combination of Independent Phase Informaton. *Acta Cryst.*, **B26**, 136–143.

Jones, Geraint. 1989. *Deriving the fast Fourier algorithm by calculation*. Technical report TR-4-89. Oxford University Programming Research Group.

Jones, T. A., Zou, J-Y., Cowan, S. W., & Kjeldgaard, M. 1991. Improved Methods for Building Protein Models in Electron Density Maps and the Location of Errors in these Models. *Acta Cryst.*, **A47**, 110–119.

Kabsch, W. 1976. A Solution for the Best Rotation to Relate Two Sets of Vectors. *Acta Cryst.*, **A32**, 922–923.

Kabsch, W. 1988. Evaluation of Single-Crystal X-ray Diffraction Data from a Position-Sensitive Detector. *J. Appl. Cryst.*, **21**, 916–924.

Kabsch, W., & Sander, C. 1983. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, **22**, 2577–2637.

Kartha, G., & Partharasarathy, R. 1965. Combination of Multiple Isomorphous Replacement and Anomalous Dispersion Data for Protein Structure Determination. I. Determination of heavy-Atom Positions in Protein Derivatives. *Acta Cryst.*, **18**, 745.

Kendrew, J. C., Bodo, G., Dintzis, H. M., Parrish, R. G., Wyckoff, H., & Phillips, D. C. 1958. A Three-dimensional Model of the Myoglobin Molecule Obtained by X-ray Analysis. *Nature (London)*, **181**, 662–666.

Kleywegt, G. J., & Jones, T. A. 1996a. *Acta Cryst.*, **D52**, 826–828.

Kleywegt, G. J., & Jones, T. A. 1996b. Efficient Rebuilding of Protein Structures. *Acta Cryst.*, **D52**, 829–832.

Kraulis, P. 1991. MOLSCRIPT: a Program to Produce Both Detailed and Schematic Plot of Protein Structures. *J. Appl. Cryst.*, **24**, 946–950.

Kraut, J., Sieker, L. C., High, D. F., & Freer, S. T. 1962. Chymotrypsinogen: a Three-dimensional Fourier Syntheis at 5Å Resolution. *Proc. Nat. Acad. Sci.*, **48**, 1417–1424.

Lamzin, Victor S., & Wilson, Keith S. 1992. Automated Refinement Procedure. *In:* (Dodson *et al.*, 1992).

Laskowski, R. A., Arthur, M. W. Mac, Moss, D. S., & Thornton, J. M. 1993. PROCHECK: a program to check the stereochemical quality of protein structures. *J. Appl. Cryst.*, 283–291.

Lattman, E. E., & Love, W. E. 1970. A Rotational Search Procedure for Detecting a Known Molecule in a Crystal. *Acta Cryst.*, **B26**, 1854–1857.

Leslie, A. G. W. 1988. A Reciprocal Space Algorithm for Calculating Molecular Envelope using the Algorithm of B. C. Wang. CCP4 Daresbury Study Weekend, nos. DL/SCI/R26, ISSN 0144–5677. Warrington WA4 4AD, UK: Daresbury Laboratory, for Daresbury Laboratory.

Luger, Peter. 1980. *Modern X-ray Analysis on Single Crystals*. Berlin, New York: Walter de Gruyter.

Machin, P. A. (ed). 1985. *Molecular Replacement*. CCP4 Daresbury Study Weekend, nos. DL/SCI/R23, ISSN 0144–5677. Warrington WA4 4AD, UK: Daresbury Laboratory, for Daresbury Laboratory.

Machin, P. A., Campbell, J. W., & Elder, M. (eds). 1980. *Refinement of Protein Structures*. CCP4 Daresbury Study Weekend, nos. DL/SCI/R16, ISSN 0144–5677. Warrington WA4 4AD, UK: Daresbury Laboratory, for Daresbury Laboratory.

Matthews, B. W. 1966. The Determination of the Position of Anomalously Scattering Heavy Atom Groups in Protein Crystals. *Acta Cryst.*, **20**, 230.

Matthews, B. W. 1968. The Solvent Content of Protein Crystals. *J. Mol. Biol.*, **33**, 491–497.

Mc Ree, Duncan E. 1993. *Practical protein crystallography*. Academic Press.

Mukherjee, A. K., Helliwell, J. R., & Main, P. 1989. The use of MULTAN to Locate the Positions of Anomalous Scatterers. *Acta Cryst.*, **A45**, 715–718.

Murshudov, G. N., Vagin, A. A., & Dodson, E. J. 1997. Refinement of Macromolecular Structures by the Maximum-Likelihood Method. *Acta Cryst.*, **D53**, 240–253.

Navaza, J. 1993. On the Computation of the Fast Rotation Function. *Acta Cryst.*, **D49**, 588–591.

Navaza, J. 1994. AMoRe: an Automated Package for Molecular Replacement. *Acta Cryst.*, **A50**, 157–163.

Otwinowski, Zbyszek. 1991. Maximum likelihood refinement of heavy atom parameters. *In:* (Wolf *et al.*, 1991).

Otwinowski, Zbyszek. 1993. Oscillation data reduction program. *In:* (Sawyer *et al.*, 1993).

Perutz, M. F. 1956. Isomorphous Replacement and Phase Determination in Non-centrosymmetric Space Groups. *Acta Cryst.*, **9**, 867.

Perutz, Max. 1992. *Protein structure: new approaches to disease and therapy*. New York: W.H. Freeman and Co.

Podjarny, A.D., Bhat, T.N., & Zwick, M. 1987. Improving Crystallographic Macromolecular Images: the Real-space Approach. *Ann. Rev. Biophys. Biophys. Chem.*, **16**, 351–373.

Press, W. H., Flannery, B. P., Teukolsky, S. A., & Vetterling, W. T. 1986. *Numerical Recipes: The Art of Scientific Computing*. Cambridge Scientific Press.

Priestle, John P. 1988. RIBBON: a stereo cartoon drawing program for proteins. *J. Appl. Cryst.*, **21**, 572–576.

Ralph, A. C., & Woolfson, M. M. 1991. On the Application of One-Wavelength Anomalous Scattering. III The Wilson and MPS Method. *Acta Cryst.*, **A47**, 533–537.

Raymond, Eric (ed). 1993. *The New Hacker's Dictionary*. Second edn. MIT Press.

Read, R. J. 1986. Improved Fourier Coefficients for Maps Using Phases from Partial Structures with Errors. *Acta Cryst.*, **A42**, 140–149.

Read, R. J. 1991. Dealing with imperfect isomorphism in multiple isomorphous replacement. *In:* (Wolf *et al.*, 1991).

Rhodes, Gale. 1993. *Crystallography Made Crystal Clear: A Guide for Users of Macromolecular Models*. San Diego; London: Academic Press.

Rossmann, M. G. 1960. The Accurate Determination of the Position and Shape of Heavy-Atom Replacement Groups in Proteins. *Acta Cryst.*, **13**, 221.

Rossmann, M. G. 1961. The Position of Anomalous Scatterers in Protein Crystals. *Acta Cryst.*, **14**, 383–388.

Rossmann, M. G., Arnold, E., & Vriend, G. 1986. Comparison of Vector Search and Feedback Methods for Finding Heavy-Atom Sites in Isomorphous Derivatives. *Acta Cryst.*, **A42**, 325–334.

Sawyer, L., Isaacs, N., & Bailey, S. (eds). 1993. *Data Collection and Processing*. CCP4 Daresbury Study Weekend, nos. DL/SCI/R34, ISSN 0144–5677. Warrington WA4 4AD, UK: Daresbury Laboratory, for Daresbury Laboratory.

Sheldrick, G. M. 1991. Heavy atom location using SHELXS-90. *In:* (Wolf *et al.*, 1991).

Singh, A. K., & Ramaseshan, S. 1966. The Determination of Heavy Atom Positions in Protein Derivatives. *Acta Cryst.*, **21**, 279–280.

Smith, David, & Howell, Lin. 1992. Identification of Heavy-atom Derivatives by Normal Probability Methods. *J. Appl. Cryst.*, **25**, 81–86.

Stout, George H., & Jensen, Lyle H. 1989. *X-ray Structure Determination: a practical guide*. Second edn. New York: John Wiley.

Ten Eyck, L. F. 1985. Fast Fourier Transform Calculation of Electron Density Maps. *In:* (Wyckoff *et al.*, 1985).

Ten Eyck, Lynn F. 1973. Crystallographic Fast Fourier Transforms. *Acta Cryst.*, **A29**, 183–191.

Ten Eyck, Lynn F. 1977. Efficient Structure-Factor Calculation for Large Molecules by the Fast Fourier Transform. *Acta Cryst.*, **A33**, 486–492.

Terwilliger, Thomas C., & Eisenberg, David. 1983. Unbiased Three-Dimensional Refinement of Heavy-Atom Parameters by Correlation of Origin-Removed Patterson Functions. *Acta Cryst.*, **A39**, 813–817.

Terwilliger, Thomas C., & Eisenberg, David. 1987. Isomorphous Replacement: Effects of Errors on the Phase Probability Distribution. *Acta Cryst.*, **A43**, 6–13.

Tickle, I. J. 1985. Review of space group general translation functions that make use of known structure information and can be expanded as Fourier series. *In:* (Machin, 1985).

Tickle, I. J. 1992. Fast Fourier Translation Functions. *In:* (Dodson *et al.*, 1992).

Tickle, Ian. 1991. Refinement of SIR heavy atom parameters in Patterson vs reciprocal space. *In:* (Wolf *et al.*, 1991).

Tronrud, D. E. 1994. Methods of Minimization and their Implications. CCP4 Daresbury Study Weekend, nos. DL/SCI/R35, ISSN 0144–5677. Warrington WA4 4AD, UK: Daresbury Laboratory, for Daresbury Laboratory.

Vaguine, A. A., Richelle, J., & Wodak, S. J. 1999. *Acta Cryst.*, **D55**, 191–205.

Wang, Bi-Cheng. 1985. Resolution of Phase Ambiguity in Macromolecular Crystallography. *In:* (Wyckoff *et al.*, 1985).

Wolf, W., Evans, P. R., & Leslie, A. G. W. (eds). 1991. *Isomorphous Replacement and Anomalous Scattering*. CCP4 Daresbury Study Weekend, nos. DL/SCI/R32, ISSN 0144–5677. Warrington WA4 4AD, UK: Daresbury Laboratory, for Daresbury Laboratory.

Woolfon, M. M., & Yao, Jia-Xing. 1994. On the Application of One-Wavelength Anomalous Scattering. IV The Absolute Configuration of the Anomalous Scatterers. *Acta Cryst.*, **D50**, 7–10.

Wyckoff, H., Hirs, C. H. W., & Timasheff, S. N. (eds). 1985. *Diffraction Methods for Biological Macromolecules*. Methods in Enzymology, vol. 115. Academic Press.

Zhang, K. Y. J., & Main, P. 1990. *Acta Cryst.*, **A46**, 377–381.

Zlotnik, Fred. 1991. *The POSIX.1 Standard: A Programmer's Guide*. Benjamin/Cummings.

Program index

And thick and fast they came at last
And more, and more, and more
— Through the Looking Glass

CCP4 programs

`abs` 8, 10, Determine the absolute configuration (hand) of heavy atom substructure.
`absurd` 23
`acorn` 9, ab initio procedure for the determination of protein structure at atomic resolution.
`act` 11, 26, Analyse coordinates.
`almn` 9, 42, 65, Crowther's rotation function using FFT.
`amore` 9, 41, 42, Molecular replacement (Navaza, 1994).
`angles` 11, [Unsupported.] Bonds and dihedral angles from coordinate files.
`anisoanl` 11, Analyses of anisotropic displacement parameters.
`areaimol` 11, Solvent accessible area or area differences.
`arp_waters` 11, 59, Automated Refinement Procedure (version 5.0).
`asc2p84` 20, [Unsupported.] Converts ASCII file to PLOT84-type meta file.
`astexviewer` 12, 27, Java program for interactive 3-d display of molecular structures and electron density maps.
`axissearch` 13, 55, [Unsupported.] Changes axis and cell.
`baverage` 11, 26, Averages *B* over main and side chain atoms.
`beast` 9, Brute-force molecular replacement with Ensemble Average Statistics, Maximum likelihood-based molecular repaclement. Obseleted by `phaser`.
`bones2pdb` 10, 26, Convert `bones` output to a PDB file.
`BP3` 10, multivariate likelihood substructure refinement and phasing of S/MIR(AS) and/or S/MAD.
`bplot` 11, Plot average *B*-factors along the chain.
`bulk` 9, Bulk solvent correction for translation search and rigid body refinement steps of `AMoRe`.
`cad` 8, 13, 25, 30, 31, 42, 95, Collecting assorted reflection data.
`cavenv` 11, Calculates cavities in macromolecular structures.
`ccp4i` 12–14, 58
`ccp4mg` 10, 12
`chainsaw` 9, Molecular Replacement model truncation
`cif2mtz` 13, Program to convert mmCIF structure factors (e.g. from PDB) to MTZ.
`cif2xml` 13, Program to convert mmCIF files to XML.
`combat` 7, 23, 24, Produces an MTZ file in multirecord form suitable for input to SCALA (replacement for ROTAPREP).
`compar` 11, [Unsupported.] Comparison of coordinates.
`contact` 11, 26, Calculates various contacts in protein structure.
`coordconv` 13, 26, 27, Interconverts various coordinates formats.
`COOT` 10, 12

cpirate 10, Statistical phase improvement
cross_validate 13, Validate harvest files for deposition.
crossec 8, Interpolates X-ray cross sections and computes anomalous scattering factors f', f''
crunch2 Direct method phase extension
Data Harvesting Manager 13
data harvesting manager tool for managing data harvesting files.
detwin 7, Detwin merohedrally twinned reflection data.
difres [Unsupported.] Compares two files of atomic coordinates in PDB format.
distang 11, 26, 42, Calculates distances and angles in protein molecule.
dm 10, 52, 53, Various density modification techniques: canned solvent flattening, Sayre's equation, histogram matching and iterative skeletonisation
dmmulti 10, A multi-crystal version of DM.
dtrek2mtz 7, Converts d*trek scalemerge output into MTZ format.
dtrek2scala 7, Convert integrated intensity and header files from d*trek into multi-record MTZ files suitable for scala.
dyndom 11, Determines dynamic domains when two conformations are available.
ecalc 9, 41, 42, Calculates normalised structure amplitudes.
extend 59, [Unsupported.] Extends the map to cover the specified volume of the unit cell.
extends 10
f2mtz 7, 13, 23, 25, 60, Converts a formatted reflection file to MTZ file.
fffear 10, Fast Fourier Feature Recognition for density fitting.
ffjoin Joining model fragments from FFFEAR.
fft 10, 30, 31, 42, 58, 59, 65, 73, In-memory crystallographic fast Fourier transformation (P1 only).
fftbig 65
fhscal 8, 30, 33, Scaling derivative to native by Kraut's method.
findncs 8, Detect NCS operations automatically from heavy atom sites.
freerflag 13, 60, Tag reflexions in MTZ file with free *R* flags.
fsearch 9, 6-d molecular replacement (envelope) search.
gensym 11, Generate sites by symmetry.
geomcalc 11, Does various geometry calculations on a molecule.
getax 9, Real space searching for rotation axis of a $D\langle n\rangle$ or $C\langle n\rangle$ multimer.
havecs 8, 27, 30, [Unsupported.] Generates Patterson vectors from atom coordinates (more-or-less replaced by vectors and gensym).
hbond [Unsupported.] Calculates possible main chain H-bonds.
helixang [Unsupported.] Distances and angles between helices.
hgen 11, Generate hydrogen atom positions for proteins.
hklplot 13, 27, Plots a precession picture from a reflection file.
hklview 7, 13, 27, Displays zones of reciprocal space as pseudo-precession images under X-windows.
icoefl 8, Scaling of multiple F_cs to F_{obs}.
ipdisp 7, Display various types of image file under X-Windows.
libcheck 11, 26, 58, Checks a monomer's description for refmac5.
loggraph 12, 19
lsqkab 11, 26, Optimise fit of atomic coordinates from two data sets (Kabsch, 1976).
madlat 7, [Unsupported.] Reads madnes output file(s), generates vector list for plotting in frodo to visualize reciprocal lattice.
makedict Converts PDB file to TNT dictionaries.

mama2ccp4 10, 25, Converts MAMA-format masks to CCP4 ones.
map2fs 10, Convert a CCP4 map to XtalView fsfour format.
mapdump 10, 25, Prints sections of electron density maps.
mapexchange [Unsupported.] Converts binary map file into ASCII one and reverse. (Obsolete—use NA4 format except for archaeological purposes.)
mapmask 10, 25, 59, Map and mask manipulation.
mapreplace [Unsupported.] Combines parts of two maps.
maprot 10, Map skewing, interpolating, rotating and averaging program.
mapsig 9, 10, 42, Print statistics on signal/noise for translation function map. Also sum, product, ratio of two maps.
mapslicer 12, 27, Section viewer for CCP4 map files.
maptona4 20, Converts binary map file to NA4 format and reverse.
matthews_coef 13, Misha Isupov's Jiffy to calculate Matthews coefficient.
mlphare 8–10, 26, 27, 31, 34, 37, 39, 52, 54, Phase calculation and refinement (Otwinowski, 1991).
molrep 9, automated program for molecular replacement.
mosflm 7, Program for processing image plate and CCD data.
mstamp [Unsupported.] Add 'machine stamp' to MTZ or map files lacking it.
mtz2various 13, 25, 60, Makes ASCII reflection files from MTZ file.
mtzdmp 13, Script for simple running of mtzdmp.
mtzdump 13, 25, 95, Displays the contents of the MTZ file.
mtzMADmod 13, Generates F+/F- or F/D from other for anomalous data.
mtzmnf 13, 95, Identify missing data entries in an MTZ file and replace with missing number flag (e.g. NaN).
mtztona4 13, 20, 91, Transforms MTZ file to transferable NA4 file.
mtzutils 8, 13, 24, 25, 91, Reflection data file utility program.
na4tomtz 13, 20, Transforms NA4 file to MTZ one.
ncont 11, Computes various types of contacts in protein structures.
ncsmask 10, Performs operations on non-crystallographic symmetry masks, e.g. before dm.
npo 12, 25, 27, 30, 31, Plot molecules end electron density maps.
oasis 8, Program for breaking phase ambiguity in OAS or SIR.
omit 10, Calculate omit-maps according to Bhat procedure.
overlapmap 11, 25, Calculate the overlap of two maps.
p842asc 20, [Unsupported.] Converts PLOT84-type meta file to ASCII.
pdb_extract 13, Programs for extracting harvest information from program log files.
pdbcur 11, Curation and manipulation tool for coordinate files.
pdbset 12, 26, 42, Various useful manipulations on coordinate files.
peakmax 11, 12, 30, 31, Search for peaks in the electron density map.
phaser 69
phaser-1.3 Maximum likelihood molecular replacement.
phaser-MR 9
phistats 13, Analysis of agreement between phase sets, and checking it against weighting factors.
pltdev 12, 27, 97, Convert PLOT84 metafiles to printer-speak.
pmf PMF pattersons search.
polarrfn 9, 41, 42, Fast rotation function in polar coordinates.
polypose 12, Superposition of multi-domain structures.
postref 8, [Unsupported.] Post-refinement of film data.

procheck 12, 15, 26, 60, Checking stereochemistry (Laskowski *et al.*, 1993). (Comprises nb, proclean, anglen, secstr, mplot, tplot, pplot, probplot.)
procheck_comp Version of procheck for related structures. (Uses rmsdev in addition.)
professs 8, Determination of NCS operators from heavy atoms.
proplot 12, Plotting and listing for procheck. (Uses pplot, tplot and probplot.)
rantan 9, Direct Method module for the determination of heavy atom positions in a macro-molecule structure or to determine a small molecule structure
RasMol 12
rdent 11, Create dictionary entries for restrain from PDB file.
rebatch 8, Alters batch numbers in an unmerged MTZ file (saves running mosflm again).
refmac 57–59
refmac5 11, 26, 57–59, Refine or idealize structures, using intensity or amplitude based least squares or -loglikelihood residuals.
reform [Unsupported.] Various on-line coordinate transformations.
reindex 13, Reindex MTZ file.
restrain 11, 57–59, Structure refinement (Driessen *et al.*, 1989)
revise 10, Program to generate normalised anomlous scattering factor magnitudes from MAD data.
rfcorr 9, Analysis of correlations between cross- and self-rotation functions.
rotamer 12, List amino acids whose side chain torsion angles deviate from Richardson's Penultimate Rotamer Library.
rotaprep [Unsupported.] Produces multirecord MTZ file from foreign formats.
rotgen 7, Simulate X-ray diffraction rotation images.
rotmat 9, Interconverts CCP4/merlot/X-PLOR rotation angles.
rsearch 9, 13, 42, *R*-factor search.
rsps 8, 31, 35, Heavy atom positions from derivative difference Patterson maps.
rstats 8, Scales two data sets.
rwcontents 12, 55, Count atoms by type in PDB file.
rwdict 26, [Unsupported.] PDB ↔ PROTIN dictionary inter-conversion.
sapi 8, 9, Heavy atom site location program.
sc 12, Program to analyse shape complementarity of molecular interfaces.
scala 7, 8, 24, 93, Scaling together multiple observations of reflections (replacement for ROTAVATA and AGROVATA).
scaleit 8, 30, Various derivative and native scaling.
scalepack2mtz 7, Converts merged scalepack output into MTZ format.
sfall 10, 31, 42, 65, 98, Structure factor calculations using FFT.
sfcheck 12, Program for assessing agreement between atomic model and X-ray data
sftools 13, Reflection data file utility program including some density map handling
sigmaa 8, 31, 53, 54, 58, 59, Phase combination (Read, 1986).
sketcher 11, Monomer library sketcher and graphical interface to libcheck.
solomon 10, Modifies the electron density maps by averaging, solvent flipping and protein truncation.
sortmtz 13, 18, 23–25, Sorting MTZ file(s).
sortwater 12, 26, Assign waters to nearest protein atoms.
stereo 13, Reconstruction of 3D coordinates from measurements of a stereo diagrams.

stgrid 9, Generates stereographic projection from polarrfn for measuring angular coordinates.
stnet 9, Stereographic net for use with polarrfn plots.
superpose Secondary structure allignment.
surface 12, Accessible surface area.
symfit [Unsupported.] Fit best molecular transformations to sets of crystallographic coordinates e.g., heavy atom coordinates, related by non-crystallographic symmetry.
tffc 9, 41, 42, 49, Translation function.
tlsanl 11, 58, TLS analysis after restrain
topdraw 13, Sketchpad for creating protein topology diagrams.
topp 12, An automatic topological and atomic comparison program for protein structures.
tracer 13, Reduced cell calculations.
truncate 8, 24, 55, 93, Converts intensities to amplitudes (French & Wilson, 1978).
undupl 7, Remove duplicates from madnes data, after absurd.
unique 8, 24, 60, Generate unique reflection data set.
uniqueify 8, 60, Script to complete dataset and add free-R flags.
vecref 8, 31, 37, Vector space refinement of the heavy atoms. Consider also mlphare.
vecsum 8, 31, 35, [Unsupported.] Automated Patterson solution.
vectors 8, 27, 30, Generates Patterson vectors from atomic coordinates.
volume 12, Polyhedral volume around selected atoms.
watertidy 12, 26, Rearranges water position.
watncs 12, Pick waters which follow NCS and sort out to NCS asymmetric unit.
watpeak 12, Selects peaks from peakmax and puts them close to the respective protein atoms.
wilson 8, 24, Makes Wilson plot.
xccpjiffy2idraw 12, Converts xloggraph/xplot84driver output to (editable) idraw format.
xdldataman 13, manipulates reflection files, exchanges formats etc.
xdlmapman 10, 25, 59, manipulates maps, exchanges formats etc.
xloggraph 13, 19, 23, 53, 70, 74, 86, Display plots derived from log files under X-Windows.
xplot84driver 12, 27, 30, Display PLOT84 plots under X-Windows.
zeroed [Unsupported.] Sets part of the map to zero.

Non-CCP4 programs

BP3, 9
Babel, 13, 80
CNS, 9, 11, 13, 31, 42, 57
FRODO/TOM, 80
HySS, 9
LAUE, 7
MAMA, 10, 25, 26
MERLOT, 9, 42
MULTAN, 8, 9, 36
OOPS, 10
O, 10, 12, 25–27, 31, 59, 81
PATSEE, 81
PRISM, 52
PyMOL, 10
RAVE, 10, 81
SHARP, 8–10
SHELX-97, 8, 9, 11
SHELX86, 36
SHELX93, 57
SHELX, 13, 31, 60, 81
TNT, 11, 31, 57
Tcl/Tk, 73
X-PLOR, 9, 13, 26, 60
XDS, 7, 23

angel, 10
archie, 82
arp_warp, 11, 58, 59
bash, 18, 71
bibtex, 117
blt, 73
bones, 10
cd, 73
chemnotes, 26
configure, 71–76
corels, 80
cp, 75
csh, 18, 71, 72
d*trek, 7
demon, 10, 80
denzo, 7, 23
diff, 73
dssp, 11, 80
duptree, 71
dvips, 117
f2c, 74, 82
frodo, 12, 31, 96
ftnchek, 87
ftp, 79
g77, 82
gcc, 74, 82, 85
gunzip, 73
heavy, 37
html, 69
idraw, 12
idxref, 93
install, 75
ksh, 18, 71
ld, 75
ln, 75
lynx, 86
m4, 75
madnes, 7, 23, 80
madsys, 10, 39, 80
makeindex, 117
make, 71–73
man, 70, 72, 86, 113
mapbrick, 96
mapcont, 96
merlot, 42, 81
molscript, 12, 27, 81
mosflm, 8, 23, 24, 81
multan, 31
noweb, 89
ortep2, 12
oscgen, 93
p2c, 85
pacman, 42
patch, 73
patsee, 31
pdbrot, 12
pltout, 12
postplot, 12
prepi, 12
ranlib, 75
raster3D, 12, 81
ribbon, 12, 27, 117
ribrot, 12
setor, 12
sftools, 81
sh, 18, 71
splitd, 12
strip, 73
tar, 70, 72
tcsh, 18, 71
weis, 81
what-if, 60
wulff.ps, 13
xentronics, 7
xfig, 117

Index

Yes, this index is incomplete and untidy…

`man`, 72

ab initio phases, 8
absorption corrections, 23
abstraction, 85
Adams, Douglas, 63
aggregated software, 7, 12, 13, 70
Alice in Wonderland, ix, 82
analysis of coordinates, 7
anomalous differences, 94
anomalous occupancy, 37
anomalous scattering, 8, 108
anonymous ftp, *see* ftp
archaeology, 109
area detector, 5, 7
asymmetric unit, changing, 8
atomicity, 52
averaging, 52, 54

Ballard, Charles, 1
binary files, 20
Bionet, 81
BMCD, 81
Briggs, Peter, 1
Brookhaven format, 26
bugs, 77
    reporting, 78, 80
bulk solvent, 8
bulk solvent correction, 58
'bulletin board', 80

C, 2, 69, 85, 89
C++, 2, 69
Campbell, John, ix
Carroll, Lewis, ix
CCD, 7
ccp4i, 22, 73
centricity, 94
CIF, 97
citation, CCP4, 2
column labels, MTZ, 94
column types, MTZ, 93
combination of data, 8
command line arguments, 15, 20, 22
common error, 19
common keywords, 17
condition code, 19
configuration, 71
connectivity, map, 52
constitution, CCP4, 1
coordinate transformations, 110
coordinates
    interpretation and manipulation, 26
copyright, 87
Cowtan, Kevin, ix
cross rotation function, 6, 42
cross vectors, 44
cross-validation, 60
crystallisation database, 81
crystallographic software, non-CCP4, 80
Cygnus Support, 78

Daly, Peter, ix
data collection, 5, 7
data combination, 8
data formats, 86
data reduction, 7
DCL, 69, 89
density maps, 6
density modification, 51, 108
density,protein, 55
derivative data, 8
difference map, 59
diffractometers, 5, 7
direct methods, 31
distributed file system, 75
documentation, 15, 85, 86
    metalanguage, 16
Dodson, Eleanor, ix

e-mail, 77
Easter egg, 117
environment variable, 15, 16, 20
error messages, 19

Eulerian angles, 42
Evans, Phil, ix, 1
examples, 69, 72

FAST, 7
fast Fourier transform, *see* FFT
FFT, 10, 63, 88
file
    extension, default, 16
    extensions, 21
    library, 16
file formats, 1, 2, 77
files
    connexion, 15
    input and output, 15
    pre- release 2.2, 20
film, 5, 7, 8
Fortran, 2, 69, 86, 89
    FORTRAN77, 1, 85, 89
    Fortran90, 85
    non-portable features, 86
    standard, 85
fractional coordinates, 26
fractional sites, 8
free *R* factor, 60
    density modification, 53
free software, 2, 63, 81
Free Software Foundation, 81
ftp, 70, 77, 79, 80, 87
    sites, 81
funding, 1

C. F. Gauß, 63
geometric corrections, 23
GNU, 73
GNU project, 73, 81
Graphical user interface, 2
graphical user interface, 22
graphics
    editing, 12
graphics metafile, 97

heavy atom positions, 5, 32
Hendrickson-Lattman coefficients, 40, 94
Henrick, Kim, ix
histogram matching, 10, 51, 53, 108
homologous protein, 6
Howard, Maeri, 1
HPGL, 12

image plate, 5, 7
initialisation
    program, 86
input
    keyworded, 15
IRIX, 70, 75
isomorphous differences, 8

Jane Richardson-type cartoons, 12

Kabsch algorithm, 26
Keegan, Ronan, 1
keyworded input, 15, 16
keywords, 86, 88
kluges, 1–117
kōan, 117
Kraut scaling, 33

Laue method, 7
Leslie, Andrew, ix
library routines, 85
Linux, 2
literate programming, 88
log file, 15, 19, 78
logical names, 15, 20, 86, 89
    common, 16
    library, 16
login procedure, 15
Love, Dave, ix, 117
Lp correction, 23

machine stamp, 20, 91
MAD, 10, 37, 38, 52
map files, 10, 20
maps
    calculation, 10
    extension, 10
masks, 96
Matthews' number, 55
maximum likelihood, 6, 34, 58
maximum likelihood phasing, 31, 36
Mc Laughlin, Sandra, ix, 91
merging, 5, 8, 23
metalanguage
    documentation, 16
Miller indices, 19
MIR, 30, 52
missing data, 93
molecular dynamics, 6
molecular envelope, 6
molecular replacement, 6, 9, 41, 52, 53

molecular weight, 55, 56
MS Windows, 12
MTZ, 17, 91
    header, 17
MTZ files, 20
    column labels, 94
    column types, 93
    multi-record, 20
MTZ format, 19
multi-record files, 7, 20, 23, 24, 92

NA4 format, 91
Naismith, Jim, 1
NaN, 95
NCS averaging, 10
Netlib, 87
network news, 81
noise filtering, 54
non-crystallographic symmetry, 6, 42, 45–46, 52
    refinement, 53
numerical code, 87

Onesti, Silvia, ix
operating system dependencies, 89
operating systems, 77, 89
orthogonal coordinates, 26
outliers, 23
output
    verbosity, 22

partiality, 94
partially recorded reflexions, 8
Pascal, 85
patch files, 77
patents, 63
Patterson, 31
Patterson functions, 33
Patterson map, 5, 6, 8
Patterson search, 8
Patterson vectors, 8
PC, 14
peak search, 8
phase combination, 8
phase determination, 38
phase extension, 52, 54
phase improvement, 10, 53, 54
phase refinement
    maximum likelihood, 36
phased translation function, 42
Photon Factory, 81
pitfall, 16
PLOT84, 88, 97
PLOT84, 98
plotting, 7, 12, 13, 25, 27
portability
    code, 85, 86, 89
    data files, 20
porting, 77
POSIX, 85, 89
postrefinement, 8
PostScript, 12, 15, 70
pre-processor, 87
precession pictures, 27
program initialisation, 86
program options, 22
program output, 19
program startup, 21
program termination, 86
python, 69

R factor search, 6
Ramachandran plot, 11, 12, 60
references, 2
refinement, 5
    Hendrickson–Konnert, 57
    least-squares, 57
reflexion files, 19
relational database, 19, 91
release policy, 77
Remacle, Francois, 1
reporting bugs, 78
ribbon diagrams, 12, 27
Richard Stallman, 78
rigid body fitting, 41
rigid body refinement, 6, 42
Rolfe, Daniel, 1
Rop, 95
rotation function, 42–46
Rypniewski, Wojtek, ix

Sayre's equation, 10, 52, 53, 108
scaling, 5, 7, 8, 23
scratch files, 21
self rotation function, 6
self vectors, 44
shared libraries, 72
shell, login, 71
shells, Unix, 18
Sim weight, 8

Skarzynski, Tadeusz, 1
skeletonisation, 10, 52, 108
small molecule drawing, 12
solvent content, 55
solvent flattening, 6, 10, 51, 53
sorting, 13, 25
spacegroups
    specifying, 18
standard input, 15
standard output, 15
standards
    C, 85
    Fortran, 85, 87
    POSIX, 85
stealing code, 87
Steele, Guy L., 117
Stein, Norman, 1
stereochemistry, 12
subroutine library, 1
support, 77
symmetry equivalents, 23
symmetry operators, 18
syntax description, 16

Tcl/Tk scripting language, 22
Tektronix, 12
temperature factor, 8
termination
    program, 86
Terry, Howard, 91
Tickle, Ian, ix
TLS refinement, 57, 58
Toolpack, 87
tools
    Fortran, 87
translation function, 6, 42, 47–49
    phased, 42
translation search, 46–47

unique reflexions, 8
Unix, 1, 2, 15, 16, 19–21, 69, 71, 77, 82, 86, 89
    common error, 19
    pitfall, 16
    shells, 18
unmerged data, 20
update notifications, 77
URL, 80
Usenet, 18, 82

verbose output, 22
version numbers, 77
VMS, 15, 77, 86, 88, 89

web site, 69, 70
Wilson plot, 8, 24
windows, 89
Winn, Martyn, 1
Wolf, Wojtek, ix
World Wide Web, 81
writeups, 15
WWW, *see* World Wide Web

X-Windows, 7, 12–14, 27, 70, 71, 74, 111
Xentronics, 7

Zelinka, Jan, 91

Colophon[4]

This manual was typeset using LaTeX in (mainly) Palatino, Lucida Bright Sans, Computer Modern Typewriter and Lucida New Math PostScript fonts rendered by `dvips` and printed at 600 dpi on a NEC Silentwriter S102P. Various common style options and special definitions were used—see the source. The bibliographic information was produced using `bibtex` and the index using `makeindex`; some figures were drawn with `xfig` and converted to PostScript, which was included with the `psfig` macros. Since the editor considers himself a competent LaTeX hacker, the source might serve as a useful model for producing similar documents such as theses.... The bibliography file (in BibTeX format) might also be useful.

And to answer the question on everyone's lips: the structure on the front cover is the DHFR example from the `ribbon` distribution.

[4] Look it up!