Eden User Manual
Version 5.2

Hanna Szőke

September 22, 2004

Contents

Chapter 1

Introduction

Lack of information cannot be remedied
by any mathematical trickery. *Lánczos*[10]

The computer program Eden (short for Electron density) is based on a real-space method for crystallography. The origin of the method came from the analogy between holography and the problem of recovering the electron density of molecular constituents of a crystal. It reins in the techniques of image processing to aid in the search for a crystal structure.

The most unconventional feature of the holographic method is that it is a real-space method: it searches for a distribution of electrons in the unit cell that meets all the known constraints on the molecules themselves, while giving rise to the observed diffraction pattern. The phases are thus free to change (within certain limitations). This manual gives short shrift to the theory of the holographic method. However, it is important that as a potential user of the program, you become familiar with that theory. For an overview of the theory, see [12]. More detailed information is in [13] and [11]. Recent papers, [14] and [15], give further details. We urge you to read [12] before you attempt to run Eden.

It may be helpful to explain from the outset what Eden does *not* do. It does not deal in atoms at all: except for one recently-added module, it neither reads nor writes pdb files (except in very trivial circumstances), it has no knowledge of chemical bonds or valencies, let alone amino acids or helices. However, that is not to say that it cannot determine the positions of atoms! It may provide a well-circumscribed volume within the unit cell which can easily be identified as a sulphur atom, for example, when viewed under O, but the word "sulphur" will not appear in the Eden output. Essentially, the only piece of chemical information that the program has and uses very effectively is that electrons cannot have negative density! The connections of Eden to the world of conventional crystallography are via the structure factors – fobs measurements that are read and fcalc models that are read and written – and map files for visualization that are written.

Eden is capable of solving crystals containing macromolecules (proteins, RNA or DNA) of current interest to biologists and biochemists Recently, physicists have shown an interest in using Eden for very high-resolution work with inorganic crystals as well. Eden's main advantages are that it has less bias toward its input model than usual methods and is capable of incorporating additional information in a consistent and optimal way. The program runs in a time of order $NlogN$ and it needs storage of order N, where N is the total number of resolution elements, which is about the number of reflections collected. Eden is essentially scale-independent: it is equally capable of finding single atoms in a cubic crystal measuring 5 Ångstrom to a side, as it is in identifying a protein subunit in a 500 Ångstrom crystal of the Ribosome – always, given adequate resolution and accuracy of the measurements.

Eden is written in standard C. It has run successfully on a variety of workstations in the Unix environment: SUN Sparc stations, Silicon Graphics Iris and Indy Irix, IBM RISC System/6000 Model 550, HP9000 and DEC alpha, and also under Linux on a MacOS X.

Eden consists of the actual solver program (Solve) plus an extended number of utility programs, all of which are included under a single main controller. A general description of the solver, using Crambin as an example, is described in Chapter 2. Chapter 3 deals with files: input, intermediate and output. Chapter 4 returns to the solver, with a more detailed description appropriate for realistic runs. Chapter 5 revisits the solver for a discussion of runs with multiple isomorphous replacement (MIR) and multiple anomalous dispersion (MAD). The same program, Solve, is used in these cases. Chapter 6 discusses the available physical-space constraints that may be applied in Solve. Chapter 7 discusses the available reciprocal-space constraints that may be applied in Solve. Chapter 8 turns to the preprocessors: Apodfc, Apodfo, Back, Expandfc, Expandfo, Maketar, Perturbhkl and Sym. Some of these will have to be run before you can solve any real problem. Chapter 9 describes postprocessor, Regrid. Chapter 10 reviews evaluation utilities: Dphase, Distance, Count, Shapes and Variance. Chapter 11 includes some details for fine-tuning runs, invoking other utilities, running the latest addition to Eden's modules – Refine and experimenting with the source code. Appendix A gives instructions for installing Eden; and Appendix B describes scripts for handling multiple jobs.

Version 5.2 differs from earlier versions in that it is controlled within GROMACS. Also, it is able to work in two modes: using floating-point arithmetic throughout or using double-precision arithmetic throughout. While the double version is undoubtedly more precise, size and even time considerations make the possibility of using floating-point arithmetic quite attractive. The choice is determined by a single flag in the Makefile. Version 5.2 contains a new module, Refine, that will allow you to fine-tune your .pdb file on the basis of a reliable Eden map. Please note:

> *Even if you are familiar with Eden, please check out Appendix A for*
> *instructions on re-installing the FFTW package and fine-tuning the Makefile.*

Other changes include parsing of pdb files without using a special AWK script; and handling of .fobs files that include phase information. Both fobs and fcalc files are now written out in a format that enables them to be read by other standard crystallographic programs. You no longer need to explicitly set a shell variable named $EDENHOME (the full directory path that ends with `EDEN` or `eden`); this is done automatically during the 'make' procedure. The versions in the 5 series also differ from earlier ones in the following ways: A more up-to-date version of FFTW is used. The current version saves the fft "wisdom", enabling reusage in runs within the same directory. This has two advantages: (a) multiple runs under identical conditions give bit-for-bit agreement, which is not always the case when there is no such available wisdom file; (b) the time spent by EDEN setting up the fft parameters for a repeat run is essentially instantaneous, in place of a time of order 1 minute. However, we are still using version V2 of the FFTW package.

The electron density histogram information in Solve and Back is now given a sub-decade breakdown whenever one decade contains $> 50\%$ of the data. In Solve there is a high-resolution input option for accurately recovering sharp peaks in the electron density. In Solve, the fcalc is generated internally and automatically from its physical-space counterpart. Generally, Expandfo and Expandfc need not be used, since expansion to P1 is handled internally by programs reading fobs and fcalc files. There are modules for phase extension, singlet and triplet invariants and for amplitude and intensity detwinning (see Chapters 6 and 7).

Code for scaling the fobs to fcalc files has been incorporated into the apodization procedure. The scaling factor for fobs to fcalc, identified by keyword FSCALE, is now a mandatory input parameter to Solve. Non-crystallographic symmetry has been withdrawn as a constraint. Regarding the modes of operation of Solve (completion or correction), correction is now the default mode. Correction mode is generally not recommended. There are several name changes among the input parameters and a few changes in the default values of parameters. For example, by default, sigma values in the fobs file *are* now used. There are new tools for comparing the results of runs that should be similar (e.g., Solve runs of variously perturbed starting models). For all these reasons, even if you are already familiar with Eden, you should nevertheless check out this manual or the help files before resubmitting old input or using Eden for new problems.

Chapter 2

General Operation of Eden

2.1 How to Get Started in Eden

When you have properly installed Eden on your system (as described in Appendix A), you can run a test problem in the `example/` directory. It contains a README and four directories —

```
originals/      solve/      apodize/      back/
```

The file README contains fairly detailed information on what's here and how to use it. In particular, originals/ contains zipped files that have observed data, a "partial" fcalc, and the original .pdb file (that you will not, in fact, need for exercising Eden). The directories apodize/, back/, and solve/ (to be used in that order) contain what is needed to apodize and scale the starting files; prepare a real-space file corresponding to the partial model; and run Solve to find the missing electron density.

In each Eden run, you will type the word "eden" followed by the name of the Eden module that you wish to run (e.g., "back") followed by the name of an input file (e.g., "par", standing for the file "par.inp"). Usually, there are no further words that are required to run eden (although there may be optional flags, such as "-g" for "enable graphics"). All of this will be described more fully in this manual. Each Eden module writes out a log of the run in question, whose name is the module name with extension "log". For example, file `solve.log` is a log of the Solve run. All messages that were sent to your terminal will go to the log file as well. The log also contains a recapitulation of the run mode and parameters, information about the input files, details regarding the R factor, the range of electrons/voxel in the output file, and the time spent for the run. Other output files depend on the module being run. All of them will be discussed below.

2.2 Basic Parameters in the Input

If you examine the file `floor.inp` (see Table 2.1), you will see both familiar and not-so-familiar input parameters. A brief summary of the contents of that file follows. More exhaustive information, including parameters that have taken default values for this particular problem, will be given in Section 3.3.

Table 2.1: Contents of `floor.inp`

```
# The pound sign (#) indicates comments that are ignored by Eden.

CELL            40.802  18.519  22.379  90.  90.57  90.
SYMMETRY        P21
INPUT_RES       0.93
FSCALE          0.8   # as determined in apodfo
FO_FILENAME     ../apodize/cramb.fobs
MD_FILENAME     ../back/par_back
MODE            correction
```

• CELL. This is the usual set of unit cell dimensions — a, b and c, in Ångstroms, followed by the angles α, β and γ in degrees.

• SYMMETRY. This is the usual name of the space group, written without subscripts (e,g, P212121).

• INPUT_RES. This is the data resolution in Ångstroms. You may also use the name RESOLUTION .

• FO_FILENAME. The name of the fobs file.

• MD_FILENAME. The name of a real-space model in intermediate file format (omitting extension `.bin`) that is the starting model. Preparation of such a starting model is the job of Eden's preprocessor, Back.

• MODE. This may be "completion" or "correction", but "correction" is actually the default.

2.3 Help

Online help is available for each of the Eden programs. By typing

`eden`

you will get some general information. By typing

`eden` *program*

you will get general information about the named *program* (e.g. Solve or Back). If you invoke an Eden program incorrectly (with the wrong number of arguments), you get the same information.

If you invoke an Eden program with missing arguments – e.g.

`eden` *apodfo param_file_name*

you will be prompted to type the missing information.

If you invoke help explicitly by typing

`eden -h` *program*

you will be provided with guidance on the preparation of the input for *program*. An alphabetical list of all Eden's keywords (all the items appearing in input files), with brief explanations, may be reviewed by typing

```
eden -h keywords
```

2.4 Terminology

In order to keep matters as clear as possible, we try to reserve the word "model" for protein[1] or data derived from a pdb file or from a standard crystallographic program — i.e., data based on chemical information. Thus we may have model input structure factor files, as well as real-space models, derived from them by applying Eden's preprocessor utility, Back. On the other hand, files generated by Solve will be referred to as structure-factor solutions (in Fourier space) and real-space solutions, which may be converted to electron density maps by running them through Eden's post-processor, Regrid. When the origin of a structure factor file may be either an externally-derived model — from MLPHARE, for example — or the output of some Eden program, we will refer to it simply as an fcalc file. Similarly, when a real-space data file may be either derived from a model or generated by Solve, we will refer to it as an electron/voxel file. Note that the output of the postprocessor, Regrid is *not* an electron/voxel file, but rather a sampled electron density file in units of electrons/Å^3.

2.5 Notation

In this manual, the messages coming to your terminal and your input to the terminal are shown in `typewriter` font for verbatim input or in *italics* for symbolic input. Optional parameters are listed inside square brackets `[ ]`. Keywords are written in upper-case, although you may use either case in your `.inp` files. When the value of a keyword such as FSCALE is referred to in the text symbolically, it is called $fscale$.

In the text of this manual, names of all crystallographic software packages, including Eden itself, and the names of the Eden programs, are capitalized for clarity; when entering a command on your terminal, you may either use lower-case, e.g.

```
        eden solve floor
```

or capitalize the Eden program name, as in

```
        eden Solve floor
```

If you prefer to invoke Eden itself using upper-case E —

```
        Eden Solve floor
```

you will have to establish Eden as an alias for eden, or make the appropriate change in the Makefile (see Appendix A).

2.6 Display Programs

Certain parts of Eden need built-in display capabilities: we currently use `xmgrace`[2], for showing simple $x - y$ plots. The applications for which such a display program is needed are discussed in Section 8.1.

[1]In this manual, I frequently refer to all macromolecules as "proteins" even though RNA and DNA structures – and even inorganic crystals! – are treated on an equal footing.

[2]Copyright 1991 - 1995 Paul J. Turner

Chapter 3

Files

3.1 General Observations

There are 5 main classes of files associated with Eden:

• Standard crystallographic files.

• Eden input parameter files.

• Intermediate binary electron/voxel files.

• Log files.

• Cost files.

Each of these categories is discussed below. Please note that an Eden input parameter file *always* has the standard extension `.inp`; that extension need not be used when identifying such a file as an input argument. Similarly, intermediate bin files *always* have the same standard extension `.bin`; that extension need not be written when, for example, such a file is used as a solvent target or an electron/voxel starting point. Log files and cost files have standard extensions too (.log and .cost). On the other hand, structure factor files do *not* have standard extensions; for this reason, their names are always written out in full.

3.2 Standard Crystallographic Files

Briefly, standard crystallographic files are referred to in this manual by their usual extension: fobs, fcalc and pdb. Files with extension `.fobs` (or `.fo` or `.obs`), `.fcalc` (or `.fc` or `.calc`) and `.pdb` are used for input. For output, X-PLOR/CNS `.map` files are written by Eden's postprocessor, Regrid. A simple awk script, awk_xplor_to_xtal, to be found in Appendix B, can convert structure factor files to `.phs` files for use by XtalView. PyMOL uses X-PLOR/CNS `.map` files, but their extension must be changed from `.map` to `.xplor`.

Pdb files are not generally used directly in Eden. However, they may be invoked by the utilities Count, Refine, Regrid and Shapes, and also by Sym and Tohu. They are used for preparing the structure factors corresponding to heavy atom positions in MIR and MAD runs (see Section 5.2.3). Also, Eden's utility Tohu can be used to prepare fcalc files from pdb data, but generally other crystallographic packages (such as CCP4/SFALL) that are more sophisticated (and faster) than Tohu are preferable. See also Chapter 11 for special uses of pdb files by Eden's preprocessors and postprocessors.

Apodfc, Back, Expandfc and Dphase all read standard X-PLOR/CNS fcalc files. Solve, Apodfo and Expandfo all read standard X-PLOR/CNS fobs files. Solve and Back use data covering the full half-ellipsoid for which $h \geq 0$ in (hkl) space; if the data are not expanded to P1 but are in the upper half-ellipsoid, these programs will quietly expand the data. Also, if the input fcalc file to Back is not in the upper half-ellipsoid, Back will quietly transfer it to the desired region. Similarly, if the input fobs file to Solve is not in the upper half-ellipsoid, Solve will transform it internally. Although Eden programs require input of only a unique set of reflections, they read them all, expand them to the full half-ellipsoid for which $h \geq 0$ in (hkl) space. and verify that the expansions are consistent. Note that whenever fobs files are read, forbidden reflections are explicitly set to zero and are included in the fobs set. If certain reflections appear that are forbidden for the space group in question, Eden reports them. Since some standard programs that calculate Fcalc and Fobs are not very accurate, Solve may also complain about (and corret for) lack of Friedel symmetry.

Eden expects all files related to structure factors (including those with heavy atom information for MIR and MAD) to be formatted essentially like standard X-PLOR/CNS fcalc files. In other words, there should be a symbol such as INDX or INDE (containing at least IND) followed by values for h, k, and l, followed by another identifier and then an amplitude and (for fcalc files) a phase. For fobs files, the diffraction value should be followed by 2 further fields containing a symbol such as SIGMA (containing at least SIG) and a value for σ. Other columns are ignored[1]. No special Fortran format is required; fields are expected to be delimited by white space (spaces, not tabs). Regarding fobs files, Eden will by default use σ values. Use keyword USESIG with value FALSE if you do *not* want to use σ's. See Table 3.2. Note that if Eden finds no σ's in the input fobs files, it will quietly turn off the USESIG setting and report this in the log. Fobs information is expected to be amplitudes and their sigmas, *not* intensities and their sigmas.

Both fcalc and fobs files should have an entry corresponding to $h = 0$, $k = 0$ and $l = 0$. When preparing fcalc files, you should do the calculations out to infinitely low resolution ("infinity" in X-PLOR/CNS), in order to get the (000) reflection. As for the fobs file, you should set the (000) term to contain N_{el}, the estimated total number of electrons in the protein for the full unit cell, plus all solvent electrons, ordered and disordered. Unless you have a better estimate, use $\sqrt{0.1 * N_{el}}$ for its SIGMA value. The actual value of the fobs at (000) is not extremely critical; typically, we find that users may err by $10 - 20\%$ even, but your best ballpark number should be entered. While all structure factors in a model file are potentially useful, only those corresponding to (hkl)'s for which there is a measured fobs amplitude are actually used. Note that good very low (hkl) measurements are especially helpful for successful optimization in Solve. For the same reason, if your very low (hkl) measurements are suspect (e.g., "saturated"), you may want to exclude them

[1]A typical awk script for converting hkl files to X-PLOR/CNS format is to be found in Appendix B.

from the file entirely. Eden deals well with missing data, but incorrect data is worse than no data at all. [10]

Several Eden programs write calculated structure factor files. The main ones are listed here: Forth writes a file *name*`_forth.hkl` where *name* stands for the input electron/voxel file; Apodfc writes a file *name*`_apo`*.ext* where *name.ext* is the input structure factor file name; and Expandfc writes a file *name*`_P1`*.ext*. (For anomalous dispersion, it writes *name*`_P1plus`*.ext* and possibly *name*`_P1minus`*.ext*, where *name.ext* is the input file.) Note that Solve no longer writes a file *name*`.newhkl` where *name* stands for the input parameter file and B no longer writes a file *name*`_back.newhkl` where *name* stands for the input parameter file. if you want to know what these fcalc files look like, you should run Forth on the real-space output of Solve or Back. Two programs write revised versions of their input fobs files: Apodfo writes a file *name*`_apo`*.ext* where *name.ext* is the input fobs file name; and Expandfo writes a file *name*`_P1`*.ext* (For anomalous dispersion, it writes *name*`_P1plus`*.ext* and possibly *name*`_P1minus`*.ext*, where *name.ext* is the input fobs file.)

If you use O or Mapman for examining electron densities, you should run the postprocessor Regrid whose final output is a `.map` file — an electron density file in the X-PLOR/CNS format. If you display electron densities with XtalView, you should follow an Eden Solve by running Forth, to prepare an fcalc file corresponding to the binary output of Solve, and then running an awk script, `awk_xplor_to_xtal` to be found in Appendix B. You may then skip the Regrid postprocessing entirely.

3.3 Eden Input Parameter Files

The operation of each of the Eden programs is governed primarily by the parameters in its input file, *name*`.inp`. This file consists of a list of keywords followed by values, with no = sign required between them. You may include blank lines anywhere. You may append comments after keyword-value pairs; such comments are stripped from the input before it is used. The pound sign (#) signals the start of a comment.[2]

Keywords may be written in either lower or upper case. (In this manual, keywords are always written in upper case for greater visibility.) They may be ordered arbitrarily. It is assumed that no line of input contains more than 200 characters (including embedded blanks). Numbers are in free format, symbols are space- or tab-delimited, with no intervening commas. Keywords that are not required for a particular program are ignored. This is convenient, in that an input parameter file written for Solve may also be used for Dphase or Maketar, for example, and the superfluous input lines will not interfere with the program. However, this also means that if you misspell a keyword, the program will use the default value (insofar as there is a default). For this reason, we recommend that you check the log of Solve or Back carefully, to verify that Eden has used the values you intended. (Both Solve and Back produce log files in which the input that was ignored is listed for reference. All other Eden programs also produce log files, but the ignored information is not highlighted in them.) All Eden programs stop and complain if compulsory keywords are missing or misspelt.

[2] If you prefer some other special character, you may change the pound sign: in the header file, `util.h`, look for COMMENT_CHAR, change it and then recompile/reload the whole program.

Table 3.1 lists keywords and values required for all Eden programs. Apart from Solve and Back, most Eden programs have no other required input. Each keyword is followed by a typical value as it would be in a real input file. Descriptive and default information are written on the right-hand side of the page, with a leading # sign to indicate that they are comments. Of course, comments need not be written in your input parameter file. We now discuss the keywords from Table 3.1.

Table 3.1: Basic Input for All Eden Programs

Keyword	Example of value	description	default
SYMMETRY	P3221	# space group name	none
CELL	57.2 33.9 68.7 90 90 120	# unit cell dimensions in Å	none
		# and angles in degrees	none
RESOLUTION	2.0	# resolution in Å	none
RECORD	../myrecord	# name of run history file	history

• SYMMETRY. The value associated with this keyword is the space group name. Eden recognizes all 230 space groups. Rules for space group names are the same as in CCP4 [2] – indeed, the CCP4 file `symop.lib` is used for matching the name and for identifying symmetry operators for the space group. Names are the "short" form given in [8]; subscripts are typed as is and the overbar is typed as a leading $-$, so that e.g. $P2_12_12_1$ is typed as `P212121` and $P\bar{1}$ would be typed as `P-1`. You should use the conventional choices that correspond to the space groups with the first 230 numbers in `symop.lib`. Alternative choices such as $P112_1$ and $A2$ are not accepted by Eden. Where there is a choice, you have to use the conventional unique axis (b for monoclinic crystal systems and c for trigonals and hexagonals). For trigonal crystal systems, you have to use hexagonal rather than rhombohedral axes. Eden does not accept space group numbers. For space groups with alternate origins, please check `symop.lib` included in the `source/` subdirectory of EDEN/.

• CELL. This is the usual set of unit cell dimensions — a, b and c, in Ångstroms, followed by the angles α, β and γ in degrees. Currently, the only global restrictions on angles is that *all* are either $\geq 90°$ or $\leq 90°$. If the angles that are given for your crystal do not satify these rules, possibly CCP4's AXISSEARCH [2] program can help you. Eden checks that the input cell dimensions and angles are consistent with restrictions imposed by the space group. Eden sets the grid type (simple or body-centered) depending on the angles; if they are within $15°$ of $90°$, a body-centered grid type is used by default You may override the grid type setting by entering an explicit value for GRID_TYPE.

• RESOLUTION. This is the data resolution in Ångstroms. You may also use the name INPUT_RES , which was Eden's earlier name for this quantity.. It corresponds closely to the maximum resolution of the fobs file. Eden will use a grid whose spacing along the three dimensions is approximately $0.6 * resolution$ for a simple grid type or $0.7 * resolution$ for a body-centered grid type. The actual number of divisions along each axis is always a “nice” integer that satisfies the symmetry of the space group. The gridding resolution also determines which structure factors are to be used in the Eden run in question. Determining a value for *resolution* is discussed in Section 4.1.2.

• RECORD. Each Eden run is summarized in four lines that are written into a file of your choice or, by default, into a file named `history` in the pwd; the summary includes: the date and time at which it started; the directory from which it was run; the command line; and the outcome (success or failure). As successive Eden runs are done, new records are appended to the end of the history file. The use of the history file is intended to help you keep track of multiple Eden runs (when they were done, and from where, in which order, etc.)

Table 3.2 lists the full set of keywords and values for Solve runs without Multiple Isomorphous Replacement (MIR), or multiple anomalous dispersion (MAD). Added keywords and values needed by Solve with MIR and MAD are discussed in Chapters 5 and 6. Optional keywords and their possible values for the utilities are discussed in Chapters 8 and 9.

We now discuss the “other basic input” from Table 3.2.

• FO_FILENAME. The name of the fobs file. The relative or absolute directory path name should be used if the fobs file is not in the directory from which you run Eden. Generally, this file will *not* be the same file as your original set; see Chapter 4.

• FSCALE. This is the factor for scaling fobs data on an absolute scale. See Section 4.1.3.

• MD_FILENAME. The name of a real-space model in intermediate file format (omitting extensions `.bin`) Such a real-space model is generated by running the preprocessor, Back on the model fcalc file. This, too, is discussed in Chapter 4.

• NCONSTRAINTS. Count of physical-space (or reciprocal-space) constraints in the problem. A number in range $(0, 12)$ is permitted, the default being 0. In the following, [c] stands for a number in range 1, ... NCONSTRAINTS.

• CON_TYPE[c]. A descriptive word for the type of the c-th constraint. Legal values are: “target” for a generic target or “solvent_tar” for a solvent target or “stabilize_tar” for a protein target (there may be more than one), “phase_ext” for phase extension, “singlet” for (h, k, l) singlet invariants, “triplet” for (h, k, l) triplet invariants, or “cs” for crystal symmetry. All of these will be discussed in Chapters 6 and 7.

• RELWT_CON[c]. The relative weight to be used in the cost function for the c-th constraint.

• TA_FILENAME[c]. The name of a real-space model file in intermediate format (omitting extension `.bin`) that corresponds to the c-th target constraint. Such a real-space target may come from a variety of files (Section 4.1.5 and Chapter 6).

• WT_FILENAME[c]. The name of a real-space model file in intermediate format (omitting extension `.bin`) that corresponds to the weights associated with the c-th target. Such a real-space set of weights is generated by running Maketar (Section 4.1.5, Chapter 6 and Section 8.4).

Table 3.2: Complete Sample Input for Solve

Keyword	Example of value	description	default
	basic input for all Eden programs (see Table 3.1)		
SYMMETRY	P3221	# space group name	none
CELL	57.2 33.9 68.7 90 90 120	# unit cell dimensions in Å	
		# and angles in degrees	none
RESOLUTION	2.0	# resolution in Å	none
FO_FILENAME	../data/dat_P1_apo.fobs	# name of fobs file	none
FSCALE	0.8	# scaliing actor for fobs	none
MD_FILENAME	mod_back	# real-space model derived	
		# from an fcalc model.	none
	other basic input for Solve		
NCONSTRAINTS	1	# no. of cost func. constraints	0
CON_TYPE1	target	# description of first constraint	none
RELWT_CON1	0.1	# rel. wt. for 1st constraint	0
TA_FILENAME1	mytarget	# name of 1st Np space target	none
WT_FILENAME1	myweight	# name of 1st Np space weight	none
	uncommonly used input for Solve		
HIGHRES	TRUE	# enable high-res processing	FALSE
HRCUTOFF	15.	# highres cutoff	10.
DFDX_CRIT	0.1	# gradient drop per iteration	0.03
GRID_TYPE	simple	# “simple” or “body-centered”	see CELL
MIN_DENS	20.0	# Max. density (el/cub Å)	1000.
MAX_DENS	-0.5	# Min. density (el/cub Å)	0.
R_STOP	0.03	# R factor to terminate run	0
MODE	completion	# how to treat initial model	correction
DISCRP_FRAC	0.5	# fine-tuning stopping criterion	1.
TITLE	Data from 2/4/96	# anything	blank
USESIG	FALSE	# switch to use fobs SIGMAs	TRUE

Note that files identified by name in the input need not be in the same directory from which you run Eden; if they are not in that directory, you must give the path to them. Paths may be relative or absolute.

We now consider "uncommonly used input" from Table 3.2. More information is given in Chapter 11.

• HIGHRES. The solver will extract points that are particularly strong and handle them at a two-fold higher resolution. Such points will not be written out as part of the usual gridded `.bin` file, but will instead be written as an ASCII list `.list`. The high-resolution points will be merged into the full array of electron densities by including this keyword in the Regrid input.

• HRCUTOFF. When HIGHRES is in effect, this keyword defines the fractional level at which high- resolution processing will be enabled, relative to the average voxel in the unit cell. Thus, for example, using the default HRCUTOFF, if the average value of a voxel in the unit cell is 0.16, the high resolution points are those whose value is greater than 1.6. Since this is a "moving target", the number of such high resolution points and their locations are recalculated at each outer iteration.

• DFDX_CRIT. The factor governing the extent to which the inner loop of the solver will persist in trying to reduce the gradient of the function being optimized, before it gives up and returns to the outer loop. See also Chapter 11.

• GRID_TYPE. If any of the angles α, β or γ is greater than 105°, (as in the example in Table 3.2) or less than 75°. Eden uses a simple grid type. In this grid, electrons are represented as Gaussian blobs that are placed at regularly spaced positions starting at the $(0, 0, 0)$ corner of the unit cell and dividing the a, b and c axes of the unit cell into an integer number of subdivisions da, db, and dc. If all three angles are close to 90°, Eden can place its Gaussians on a body-centered grid type, consisting of the above-mentioned simple grid plus an intercalating grid. The intercalating grid places electrons at positions starting at $(da/2, db/2, dc/2)$ and extending up by da, db, and dc along a, b and c. In this manner, the maximum distance between neighboring points is decreased by a factor of about $\sqrt{3}/2 = 0.866$ at a cost of double the storage. For appropriate symmetry groups such as $P2_12_12_1$, this body-centered grid type is generally used. Since Eden will automatically choose the appropriate grid type, there seems to be little advantage in setting it explicitly. All further references to input files will disregard the explicit use of the keyword, but you should be aware that you can write it into *any* Eden input file.

• MIN_DENS. A lower cut-off for the density (in electron/cubic Ångstrom) used by the complex conjugate solver. Under rare circumstances, there may be a need to set this to something other than the default, 0. A simple way to avoid changing this cut-off is to increase $F(0, 0, 0)$ artificially.

• MAX_DENS. An upper cut-off for the density (in electron/cubic Ångstrom) used by the complex conjugate solver. The default is an unrealistically large number (10^{10}.). It is difficult to imagine circumstances under which you might want to change it.

• R_STOP. A (fractional) value for the overall R factor that will cause the solver to terminate a run.

• MODE. The associated string should be either "completion" or "correction". In completion mode, Solve assumes that the starting electron/voxel file in physical space represent a correct (if incomplete) model. It uses the optimization algorithm to search for missing electrons only. In correction mode, Solve makes no such assumption about the input model. In this case, Solve may *change* the starting electron model (electron/voxel file) in the optimization process — i.e., it is capable of adding, moving and removing electrons, so long as the resulting density remains everywhere non-negative. In either mode, the output of Solve is the full set of electrons/voxel, i.e., the recovered plus the initially known electrons at each grid point. We highly recommend using the default "correction".

• DISCRP_FRAC A fraction to multiply the usual discrepancy fraction, used to determine whether the solver should stop looking for a better solution. In the default, DISCRP_FRAC = 1. The solver then stops when the recovered $||Fcalc||$ fits the experimental $Fobs$ to within its standard deviation, σ, on the average. Sometimes, you may be optimistic and set DISCRP_FRAC to, say, 0.3. If you don't trust the quality of the data, you could use a value that is higher than 1.

• TITLE. Any string; it will be written into the log.

• USESIG. A switch (TRUE or FALSE) governing the usage of the SIGMA field in an fobs file. By default, the σ's are used.

3.4 Intermediate Binary Files

For purposes of retaining electron/voxel information in a compact fashion, Eden uses a binary file format identified by the suffix `.bin`. The information in this file is the voxel-by-voxel listing of the number of electrons and includes the 3 spatial dimensions of the problem plus the double grid (simple plus intercalating) where applicable.

3.5 Log Files

Each of the programs that may be invoked by Eden produces a log file whose name is `solve.log` or `apodfc.log`, etc. – i.e., the name of the program that was invoked, with the standard `.log` suffix. All messages that come to the terminal, be they informatory, warning or error, are also written to the log. There is usually added information, in particular if the verbose switch (`-v`) is in effect.

If there is already a file such as `solve.log` in the directory from which you now rerun Solve, the new log will normally be written to `solve1.log`. Up to 10 log files from a single Eden program may co-exist, with names `solve.log`, `solve1.log`, ..., `solve9.log`. This is good in that it prevents inadvertent clobbering of logs, but it can also be a nuisance if you forget that the basic `solve.log` may not be the most up-to-date! You can disable the multiplicity of log files by invoking Eden with the `-b` switch (b for batch; useful when running scripts).

Chapter 4

The Solver without MIR or MAD

4.1 Preparation of Input

In this section, we discuss setting up a real problem that has neither MIR nor MAD. The required tasks (represented schematically in Figure 4.1) are:

• problem definition

• resolution choice

• (for anomalous data) structure factor expansion to $P1$

• structure factor apodization and absolute scaling

• consistent model preparation

• (optionally, solvent target preparation)

4.1.1 Problem definition

It is presumed that you know the values to be used for *cell* and *symmetry*, which are properties of your crystal. Values for *cell* are the usual set of unit cell dimensions — a, b and c, in Ångstroms, followed by the angles α, β and γ in degrees. Eden checks that the input cell dimensions and angles are consistent with restrictions imposed by the space group. Eden is implemented for all space groups.

Regarding *mode*, if the problem is in reasonable shape, you will probably want to use (the default) correction mode. In this case, the solver makes no assumption about the correctness of the starting structure factor model file; it will change the model in the optimization process — i.e., it is capable of adding, moving and removing electrons, so long as the resulting density remains non-negative.

However if the problem is more than about 50% unknown, you may want to use completion mode. In this case, you will assume the correctness of the model, at least in an initial Eden run, and allow the solver to recover missing electrons. The starting model is not eroded. However, electrons may be added at positions where the model claimed a certain electron/voxel level.

Apart from the selection of *mode*, there are other ways in which Eden can direct the solver to maintain or change electrons in designated parts of the unit cell, e.g., by using the known part as a “protein” target, with

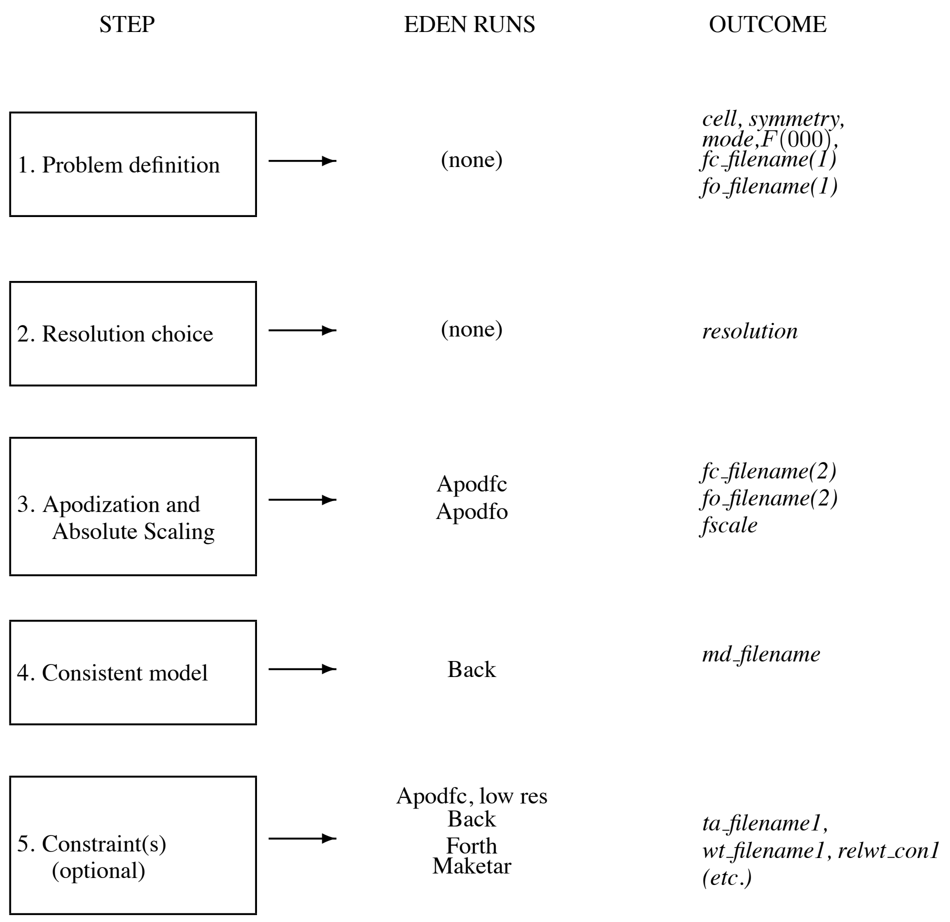
STEP
EDEN RUNS
OUTCOME
1. Problem definition
(none)
cell, symmetry, mode, F(000), fc_filename(1) fo_filename(1)
2. Resolution choice
(none)
resolution
3. Apodization and Absolute Scaling
Apodfc Apodfo
fc_filename(2) fo_filename(2) fscale
4. Consistent model
Back
md_filename
5. Constraint(s) (optional)
Apodfc, low res Back Forth Maketar
ta_filename1, wt_filename1, relwt_con1 (etc.)

Figure 4.1: Preparations for Solve without MIR or MAD

a mask around it that makes Eden maintain the part you want and leaves everything else free to change. See Chapter 6.)

Problem definition requires that you also estimate the total number of electrons in the unit cell, $F(0, 0, 0)$, including both protein and solvent (ordered and disordered). In the absence of any specific information, assume that protein has an average density of $\frac{1}{2}$ electrons/Å^3 and solvent has an average density of $\frac{1}{3}$ electrons/Å^3. Let N_p represents the number of electrons of the protein atoms in the *full* unit cell (i.e., all copies of the protein within the unit cell) and V the unit cell volume. It is easy to show that

$$F_{obs}(0, 0, 0) \approx \frac{1}{3}(V + N_p).$$

N_p can be estimated using the pdb information; it is shown in Table 4.1 that the "generic" residue has $Z_{ave} = 59.4$ electrons. Thus $N_p \simeq 60 * N_{asym} * N_{res}$, where N_{asym} is the number of asymmetric units in the crystal and N_{res} is the number of residues in an asymmetric unit. The precise value of $F(0, 0, 0)$ is not very critical to the success of Eden; it is probably best to err on the low side by about 10 – 20%. The estimate should be included as the first entry in the fobs file (a special "reflection" at $h = 0, k = 0, l = 0$) and should be accompanied by a corresponding SIGMA entry, representing the standard deviation of that value (if you plan on using σ's). In the absence of better information, use $\sqrt{0.1 * F_{obs}(0, 0, 0)}$. The fcalc file should also have an $F(0, 0, 0)$ entry: $F_{calc}(0, 0, 0) = N_p$ with a phase of 0°. If the fcalc file was calculated to "infinity" with X-PLOR/CNS, it will already contain this entry.

4.1.2 Resolution Choice

The value of *resolution* should be your estimate of the data resolution. Eden will use a grid whose spacing in the three dimensions — δa, δb and δc — is approximately $0.6 * resolution$ for a simple grid type or $0.7 * resolution$ for a body-centered grid type. The precise values of δa, δb and δc and the corresponding dimensions of the grid, N_a, N_b and N_c, are obtained as follows: the cell dimensions are divided by the desired spacing and the resulting values are rounded to the closest even product of multiples of primes less than 19. That procedure is required for the Fast Fourier Transform function used by Eden (FFTW)[1]. Additional constraints may be imposed for specific space groups: thus, for example, if the space group is $P4_1$, $P4_3$ or $P432$, N_z must be divisible by 4, and if the space group is $P3_121$ or $P3_221$, it must be divisible by 6.

It is important to remember that all the procedures in steps 3 – 7 in Figure 4.1 depend on $resolution$, so they must all be repeated if you change that resolution.

[1]http://www.fftw.org

4.1.3 Apodization, B Factors and Absolute Scaling

Apodization is surely the most unfamiliar concept that you will encounter in Eden. Remember that Eden assembles the electron density from little blobs, regularly spaced on a lattice. Now, if the real atoms in the crystal are much narrower than the blobs themselves, this sort of assembly cannot work. In fact, it is the surest way to make Eden go berserk! The recipe to avoid such a problem is to smear out the atoms to be at least as large as the blobs. In crystallographese, you have to increase the B factors of your atoms. In the more customary scientific jargon, this is called apodization (literally. “taking off its feet”)

The preprocessors Apodfc and Apodfo do this. They carry out an analysis of the structure factor data that is similar to a Wilson plot. They are used for preparing smeared versions of the “raw” fobs and fcalc files (and also, as you will see below, for determining the scale factor ($fscale$), that places the fobs on an absolute scale).

The inputs to Apodfc and Apodfo are identified by the suffix (1) and the smeared versions are identified by the suffix (2) in Figure 4.1. Insofar as they determine that apodization *is* required, Apodfc and Apodfo write smeared files whose names are derived from their input fcalc or fobs file name by appending `_apo` to the base name (to the left of the `fobs` or `fcalc` extension). Using an input named `mymod.fcalc`, Apodfc would write a file named `mymod_apo.fcalc`.

Apodfo and Apodfc read structure factors from an input fobs or fcalc file; they average the squared amplitudes, $\|F\|^2$, within shells of equal thickness in a space of $1/d^2$, where F stands for F_{obs} or F_{calc} and

$$1/d^2 = (h^2/a^2) + (k^2/b^2) + (l^2/c^2)$$

(or its generalized form for non-orthogonal crystals [6]).

Calling the shell averages $< \|F\|^2 >$, the programs prepare $ln(< \|F\|^2 >)$ as a function of $1/d^2$. They then find the slope of that (very roughly) linear function. They use two methods for deriving the slope: one is a straightforward least- squares minimization; the other more sophisticated method uses a “universal” protein correction factor [3] that suppresses much of the non-linearity. If you run the apodization programs with the `-g` switch enabled, graphs using both methods are presented for your inspection (under Xmgr) and we also print out our recommendation in the terminal report – but you may make your own choice. This is discussed at greater length in Section 8.1.

Apodfc and Apodfo determine the appropriate factor (δ_{fo} or δ_{fc}) to be used for smearing the experimental data. They use δ_{fo} or δ_{fc} to write out the apodized file (insofar as the factor is greater than zero; otherwise, you should use the original unapodized file.) Once the apodized file has been written, you need not worry about the particular δ_{fo} or δ_{fc} used; it is no longer needed as input to Solve. The actual process of apodization is quite critical to the success of Eden's solver and the fitting to determine the slope is a non-trivial procedure. For these reasons, we strongly urge you to inspect the Wilson-like plots and to read the detailed information on apodization in Section 8.1. Note too that the apodization of fobs data uses σ values (insofar as they are present) unless you turn off the USESIG switch.

Next we consider scaling, which is usually done as a part of Apodfo. *It cannot be stressed too often that all structure factors used in Eden must be on an absolute scale. In our experience, careless scaling is the one most common cause of poorly resolved electron density in Eden.*

The relationship upon which all scaling is based may be written in the form

$$< ||F_{\overline{h}}||^2 >= \sum Z_i^2 e^{-B/4d^2}$$

or

$$ln(< ||F_{\overline{h}}||^2 >) = ln(\sum Z_i^2) - B/4d^2,$$

where $F_{\overline{h}}$ is the absolutely scaled structure factor corresponding to $\overline{h} = (h, k, l)$, Z_i is the number of electrons for the i-th atom, B is an average B-factor, and

$$1/d^2 = (h^2/a^2) + (k^2/b^2) + (l^2/c^2)$$

(or its generalized form for non-orthogonal crystals [6]). Thus the graph of $ln(< ||F_{\overline{h}}||^2 >)$ as a function of $1/d^2$, called a Wilson plot, should ideally be a straight line and, if the structure factors are absolutely scaled, the y-intercept of that line at $1/d^2 = 0$ satisfies

$$ln(< ||F_0||^2 >) = ln(\sum Z_i^2).$$

If y_0, the y-intercept of the Wilson plot, is then measured for structure factors whose amplitudes are not necessarily scaled on an absolute scale and the value of $ln(\sum Z_i^2)$ is known, the scaling factor to be applied to those structure factors will be:

$$fscale = exp[-(y_0 - ln \sum Z_i^2)/2] = \sqrt{\sum Z_i^2/e^{y_0}}.$$

Note that the fobs data are scaled to the fcalc and not the other way around (as is the usual case in X-PLOR/CNS). The plot of $ln(< ||F_{\overline{h}}||^2 >)$ as a function of $1/d^2$ is not actually linear at either very high resolutions or very low resolutions. (This effect is corrected in true Wilson plots, but not in Eden's Apodfo or Apodfc.) At low resolutions, the solvent distorts the Apodfo plot. However, in an intermediate region, bounded (by default) by 3.5 Ångstrom at the low-resolution end and by 0.05 Ångstrom at the high-resolution end, the plot is linear enough that it may be used to estimate the y-intercept. That intercept is always reported as part of the output of Apodfo and Apodfc. The bounding resolutions may be changed as part of the input to Apodfc and/or Apodfo.

How should you obtain a value for $fscale$? After running Apodfo (or Apodfc), a file with extension `_wil` contains the Wilson-like plot of the apodized fobs (or fcalc) data; if two such files – one for the fobs and one for the fcalc – are already correctly scaled, those plots should essentially coincide over a fair range of abscissa values and thus: $fscale = 1$, If not, one should be able to force coincidence by adding or subtracting a fixed value to the fobs plot. A mechanism for doing this using least-squares minimization exists in Apodfo. Suppose that you have first run Apodfc:

```
        eden [-g] apodfc myparam myfc.fcalc.
```

Now (regardless of whether or nor you enabled graphics), there will be a file named `myfc_wil` in the directory from which you ran Apodfc. Next, you run Apodfo;

```
        eden [-g] apodfo myparam myfo.fobs.
```

After finishing its apodization procedures, the program will ask you whether you want to scale — `Scale? - y or n`. If you answer `y`, it will request the name of the file containing fc infomation; type `myfc_wil` (possibly with a directory prefix). It will then provide you with its best-fit value of *fscale* and will write a file `myfo_wil` containing the scaled Wilson-like plot. If you enabled graphics, the two `_wil` files will also be displayed for your inspection. See also Chapter 8.

Here are also three alternative methods for scaling. (a) If you have a reasonably good model, it is fairly simple and accurate to use the intercepts reported by Apodfo ($y_{0,obs}$) and by Apodfc ($y_{0,calc}$) to calculate *fscale*:

$$fscale = exp[-(y_{0,obs} - y_{0,calc})/2].$$

This method might be used for confirming the results of the more precise scaling procedure described above.
(b) Sharp's method: If you do not have a good model, you can use the value of $ln(\sum Z_i^2)$ derived from the protein composition, as given in the pdb file, in place of $y_{0,calc}$. Table 4.1 shows $\sum Z_i$ and $\sum Z_i^2$ for each amino acid and for the "generic" protein which is an average, weighted by the relative abundances of each amino acid in proteins [4]. The data in Table 4.1 are not currently a part of Eden. However, the value of $\sum Z_i^2$ for the full unit cell, based on the pdb file (and thus excluding at least disordered water) is calculated and reported in the Eden utility Sym.
(c) Even if you do not have a good pdb file, you surely do know how many residues are in the protein and that will yield a fair estimate of $ln(\sum Z_i^2)$; use the observation (see Table 4.1) that the average value of this sum for a single "generic" residue, $\overline{Z^2}$ is 357. Thus the full sum is $\simeq 357 * N_{asym} * N_{res}$, where N_{asym} is the number of asymmetric units in the crystal and N_{res} is the number of residues in an asymmetric unit. Note: asymmetric unit, not molecule; if your crystal has non-crystallographic symmetry, you should sum over the molecules so related.

4.1.4 Consistent Model Preparation

Once the fcalc file is properly apodized, you must prepare electron/voxel files in physical space from it. This is accomplished by running Back with *fc_filename(2)* as input — see Section 8.3.

Note the naming conventions: if you run

Table 4.1: Sum of Z and Z^2 for Protein Components

Residue	$\sum Z_i$	$\sum Z_i^2$	relative abundance[4]
Ala	38	226	8.3
Arg	85	489	5.7
Asn	60	376	4.4
Asp	59	389	5.3
Cys	54	482	1.7
Gln	68	414	4.0
Glu	67	427	6.2
Gly	30	188	7.2
His	72	434	2.2
Ile	62	340	5.2
Leu	62	340	9.0
Lys	71	391	5.7
Met	70	558	2.4
Phe	78	446	3.9
Pro	52	300	5.1
Ser	46	290	6.9
Thr	54	328	5.8
Trp	98	568	1.3
Tyr	86	510	3.2
Val	54	302	6.6
Mean	59.4	357	100

```
eden back abc
```

using an input file `abc.inp`, Back will generate a file named `abc_back.bin`. The name of the input fcalc file, identified as *fc_filename(2)* in Figure 4.1 and appearing as the value associated with keyword FC_FILENAME in the input parameter file `abc.inp`, is no longer in evidence; it is no longer needed.

4.1.5 Target Preparation

The following discussion is only an *example* of Eden's capability to impose physical-space constraints. A more extensive discussion will be given in Chapter 6. This example relates to Eden's way of imposing solvent flattening. If you know which regions in the crystal are occupied by the solvent, you have a powerful tool for increasing Eden's capabilities. However, the use of solvent flattening or, as it is known in Eden, a solvent target, is optional and may not be appropriate if the location of large parts of the molecule are unknown. The description that follows can, of course, be scripted; all its parts run fairly or very fast and need little attention from you.

In order to prepare a solvent target, you will need an fcalc file corresponding to your best model, from which you have eliminated all the solvent. Obviously, the model need not be all correct — if it were, your job would be done! — but it should cover the basic volume of the full protein. The first step is to run Apodfc at a *very low* resolution (for example, set keyword APOD_RES to 7.0 Ångstrom) [2]. Then use the output of Apodfc as the FC_FILENAME value and run Back at the *regular* resolution. This provides a highly smeared version of the protein in physical space, at the same gridding resolution as your other electron/voxel files. Next, run Maketar, which prepares two binary files with fixed names: `weight.bin` and `target.bin`. The file `weight.bin` contains weights of 0 or 1, where 1 indicates a solvent point and 0 indicates a protein point. The file `target.bin` contains the target value associated with the solvent regions (typically, the electron/voxel value corresponding to $\frac{1}{3}$ electrons/Å^3). These two files should be used as is in the Solve process, where (assuming that the solvent target is constraint # 1) `target.bin` serves as the value associated with keyword TA_FILENAME1 and `weight.bin` – as the value associated with keyword WT_FILENAME1. The solver process is set up to deal with arbitrary weights in the range (0,1), with allowance for levels of uncertainty in your knowledge of the content of a voxel, but Maketar does not currently use this capability. Maketar will, by default, set roughly 50% of the unit cell to be solvent. This default may be overridden if you have reason to believe that the solvent region of the crystal deviates significantly from 50%. Maketar also allows you to set the solvent density to any value. By default, the solvent density is 0.34 electrons/cubic Ångstrom.

Finally, you must also select the relative weight to use in Solve for imposing the solvent target as an optimizing condition. You may have to try several values for this relative weight; a value less than 0.001 will probably be ineffective, while a value greater than 0.1 will probably enforce the target solvent value much too strongly giving rise to a visible "edge". For determining the relative weight, you should examine the cost function report. Typically, at least in the first outer iteration, the target contribution should be relatively small; later, it should approach the hkl contribution or even surpass it. See Section 6.5.

4.2 Running Solve: the Optimization Process

Invoke the solver by typing

```
        eden [-v] solve name
```

where *name*`.inp` contains the input parameters.

Switch `-v` is the verbose switch; it sends the running output of the cost function (which is described in the rest of this section) to a file named *name*`.cost`, for your inspection. We do recommend using this option, particularly if you have constraints; otherwise, it is difficult to assess whether the relative weights of your constraints are appropriate.

The main loop of Solve is devoted to finding an optimal set of electrons per voxel. The search, using a

[2] Another way to prepare an fcalc corresponding to the solvent region uses X-PLOR/CNS; see also Section 8.4.

conjugate gradient solver [7], is conducted in physical space; the cost function value, used for deciding how to progress in the search, has both physical-space and Fourier-space components. We first consider the general flow of control, then the cost function.

There are two levels to the iteration process — an inner loop and an outer loop. The inner loop is contained within the conjugate gradient solver which continues to search until one of a number of criteria is met. These criteria include "normal" exits: the gradient has fallen to a preset fraction ($dfdx\_crit$) of its initial value; the cost function is essentially zero; a (local) minimum in the solution surface has been found. Another reason for stopping is that the discrepancy principle was satisfied; this means that the amplitudes of the calculated structure factors fit the observed structure factor amplitudes to within their measurement error. This happens when the cost function has fallen below a minimum dictated by the σ values in the input fobs file [15]. Letting h stand for the (hkl) triplet and N_h for the number of measured structure factors, the minimum is:

$$f_{min} = \left[(N_h/2) \sum_{h=1}^{N_h} w(h)^2 \right] / \left[\sum_{h=1}^{N_h} w(h)^2 / \sigma(h)^2 \right]$$

where the weights, $w(h)$ are 1 wherever there is a data value at h, 0 otherwise. This stopping condition effectively prevents Solve from overfitting the diffraction data. You have added control over this ending condition via the discrepancy fraction, DISCRP_FRAC .

Occasionally, there are pathological end conditions. One such reason to stop the search is that a hard-wired maximum number of calls to the search function was reached. In our experience, this is a symptom that the solver is truly stuck in some local minimum.

After the conjugate gradient solver returns to the outer iteration loop with its best effort, Solve resymmetrizes the solution (see Section 4.3). It recalculates the R factor and the standard deviation between fobs and newly updated fcalc data; it writes out the current electron/voxel files; and then it applies its own criteria for continuation. If the standard deviation is not decreasing; if the changes in the electron/voxel files are essentially nil; if the R factor has fallen below a preset cut-off (r_stop); or if the discrepancy principle (see Chapter 11) is satisfied — Solve stops.

Since interim information is written out after each outer loop iteration, you may kill the Solve run (if you sense that it is not getting anywhere) without losing more than the most recent partial outer loop iteration. In fact, unless you are unlucky, an interrupt will trigger the Solve process to write out all its most current information before quitting.

It may seem that the R factors (reported as fractions, not percentages) achieved by Solve are remarkably low, but our experience has been that their significance is limited. Eden does not do conventional refinement. It does not incorporate chemical information, such as bond angles and bond lengths. Thus very low R factors may be achieved without the corresponding electron density maps being necessarily meaningful. If the number of unknowns (electrons/voxel) is much larger then the number of equations (number of reflections), the solver will always be able to overfit a "solution" for which the R factor is essentially 0.

The physical-space solutions that are calculated after each outer iteration are over-written to a file: in the example of Section 2.1, the name was `floor.bin`.

The cost function always has a Fourier-space component (which may include MIR or MAD terms). There may also be a phase extension cost function in reciprocal space. The cost function may also have one or more physical-space components, each governed by its own relative weight. The physical-space components can include one or more target cost function(s) among others. See Chapter 6.

4.3 Maintaining Crystal Symmetry

After each of the outer iterations of Solve and before writing electron/voxel arrays to disk, the arrays are symmetrized according to your space group. Differences among the electron/voxel values at symmetry-related points that exceed 10% of their average are noted and the number of such aberrant points is reported in the log. The rms fractional distance between the electron/voxel values before and after symmetrization is also noted. In a more heavy-handed way of enforcing symmetry, it is possible to use a crystal symmetry cost function and "encourage" symmetrization of the electron/voxel arrays at each internal step of the optimizer (see section 6.4).

One might expect crystallographic symmetry to be maintained without any special provisions, since internally, the fcalc and fobs files are checked and expanded to $P1$ based on the appropriate space group. In particular, forbidden reflections are explicitly set to zero and are included in the fobs set, while missing reflections that are not forbidden are not included in the optimization process. In fact, our experience is that gross crystal symmetry violations in the first outer iteration of the solver are fairly infrequent and generally represent errors in the input. There are certain exceptions to this: if your model file was prepared using a version of Sfall that is not up-to-date (say, V1.5), there may be inconsistencies in centric reflections. Also, if the data are twinned, you may see this phenomenon. See Section 7.3.
Note too that in later iterations of Solve, especially when there are spatial cost functions, numerical instability can apparently cause some violations of crystal symmetry but not gross violations.

4.4 Output from Solve

In addition to a running log named `solve.log`[3] Solve produces the following output, updated after each outer iteration:
• *name*`.bin`, containing the current solution in physical space and (if high-resolution is in effect)
• *name*`.list`.

If you run Solve with the `-v` (verbose) switch, there will be further output:

[3]or `solve`*m*`.log`, where *m* stands for the first available digit in range 1 – 9. You can check the latest log name in the history file.

- *name*`.cost`
- `outlier0`

name`.cost` contains the cost function for the native and for each constraint, recorded at each call to the function that calculates the cost. If there is MIR or MAD, the derivative costs are also recorded. The file `outlier0` contains information about those reflections whose current $||fcalc||$ differ by more than 4σ from the input $fobs$. The information contains: $d,\ nsig,\ h,\ k,\ l,\ Fobs,\ ||Fcalc||,\ and\ \sigma$, where d is the resolution of the reflection:

$$1/d^2 = (h^2/a^2) + (k^2/b^2) + (l^2/c^2)$$

and

$$nsig = (Fobs - ||Fcalc||)/\sigma.$$

It is usually most convenient to sort `outlier0` by the $nsig$ field in order to study the very far outliers. (For example, use `sort -nr +1 <outlier0 >soutlier0`.) General information about the distribution of $nsig$ among all reflections is to be found in the log, when the `-v` option is in effect. Although it is unlikely that your data will behave like a true Gaussian distribution, you may hope that, by the end of the Solve run, the percentage of far outliers will be fairly small. Note that when the solver is run with MIR or MAD, corresponding files `outlier1`, `outlier2`, etc. are written for each derivative. In this case, the value of $nsig$ is multiplied by the appropriate relative weight. Note too that if, for some reason, you are not using the σ's, the outlier report will be meaningless, since Eden will use $\sigma = 1$ everywhere. The outlier report can be useful for checking the proper scaling of experimental data and the spotting of some glitches in it.

Chapter 5

The Solver with MIR or MAD

5.1 Overview

Up to this point, we assumed that you have a single set of experimental data, you have solved your protein at least partially and you have a reasonable model which you wish to complete and correct using Eden. Now we will consider other more usual starting points for the determination of a protein structure. In addition to (or in place of) the native crystal measurement, you may have a series of measurements for 1 or more derivative protein crystals and a good idea of the positions, occupancies and B values of the heavy atoms in each of these derivatives. You may also have a starting MIR phase set, derived, for example, from PHASES, MLPHARE, SHARP or SOLVE. Additionally or alternately, you may have a series of anomalous dispersion measurements and, again, knowledge of the positions, f', f'', Z values, B values and occupancies of the wavelength-dependent anomalous scatterers. All these data set may have their own (probably lower) resolution.

In such circumstances, Eden is useful for fine-tuning a solution and for quality checking. It is worth the trouble if you want to get an electron density that optimally uses your hard-won data or if you care to know how isomorphous your derivatives are. In principle, the model you publish and deposit in the Protein Data Bank should improve by the full, optimal use of X-ray data. This, however, is very hard to quantify and we know of no documented body of experience with it.

Eden's Solve program handles problems with MIR and/or MAD data by treating native and all other structure factors on an essentially equal footing. In the optimization process, Eden minimizes a cost function which is the sum of weighted terms, one for each measured structure factor; terms are proportional to the squared differences between the calculated and measured structure factor amplitudes. Now let us assume that we have – say – one further set of MIR measurements in addition to the native ones. We can then set up a second cost function that measures the difference between two new sets of structure factors: the native calculated structure factor amplitudes are replaced by ones that include the heavy atoms, while the native measured data is replaced by data from MIR derivative crystal measurements. Eden then minimizes the native plus derivative simultaneously. For M derivatives, there are $(M+1)$ sets of equations to replace the single set of equations in our previous discussion, from which the Gaussian blobs for the native are to be found. The cost function contains $(M+1)$ times as many terms as in the simple run. Note that this procedure does assume complete isomorphism between native and derivative, except for the heavy atoms.

The main difference between the MIR and MAD cases is in the expansion of the data to $P1$, which will be discussed in Section 8.2. In this manual, we use the term "derivative" to refer to either an MIR derivative or a MAD data set. The MIR or MAD algorithm is discussed in [11] and [14]. Solve will search for the native, or (in the case of MAD) a fictitious native — i.e., the part of the molecule that has *no* anomalous scattering. Once again, we refer you to [14].

In this chapter, we will first consider the new input parameters for an MIR or MAD run and the new preparation steps that are to be added to those described in Section 4.1 for apodizing and scaling all the data sets. Section 5.2 discusses the way in which the positions and occupancies of the heavy atoms may be more precisely pinpointed, by doing a series of preliminary Solve runs at relatively low resolution.

When you have satisfied yourself that the heavy atoms are correctly described and the data sets are optimally scaled, a high-resolution run will generally yield a much improved set of electron densities. In such a high-resolution run, there is no difficulty (in fact, there is a distinct advantage) in combining MIR or MAD processing with a solvent target. In preparation for the high-resolution run, you may replace the original native model file by the best output from the preliminary runs; this improved model should serve both for purposes of preparing the solvent target and for use as a starting model in the high-resolution run. Finally, Eden may be used as an excellent check of isomorphism: take the best result from the high-resolution MIR or MAD run and use it as a starting point versus the single data set for the native and versus each of the derivative data sets separately. Insofar as the results diverge from one another, you will be able to gauge where and by how much the original crystals did not have complete isomorphism.

5.2 Preparation of Input with MIR or MAD

5.2.1 MIR and MAD Input Parameters

Input parameters for a Solve run with MIR or MAD consist of exactly the same set of input parameters as were described in Section 3.3 and Table 3.2, plus the following ones shown in Table 5.1. The table lists file names and scaling factors for 2 derivatives; this is an example only; you may have as many as 8 derivatives. These input parameter are now briefly discussed; [m] stands for 1, 2, ..., NDER:

• FO_DER_FN[m]. The name of the fobs file for the m-th derivative. (FN stands for "filename").

• FC_HEAVY_FN[m]. The name of the file containing calculated structure factors for the heavy atoms. Note that this file contains "hydrogen-like" structure factors — i.e. the value of Z is not used in their calculation. The appropriate Z, together with f' and f'' are applied within Solve.

Table 5.1: MIR or MAD Input for Solve

Keyword	Example of value	description	default
NDER	2	# no. of MIR or MAD derivatives	0
FO_DER_FN1	xglpb_apo.fobs	# 1st deriv fobs file	none
FC_HEAVY_FN1	mod1_apo.hkl	# 1st heavy atom fcalc file	none
FO_DER_FN2	xglhg_apo.fobs	# 2nd deriv fobs file	none
FC_HEAVY_FN2	mod2_apo.hkl	# 2nd heavy atom fcalc file	none
	for MAD only		
Z1	76	# At. no. 1st heavy atom	none
FP_FPP1	-17.2 20.1	# f' f'' for Z1 at 1st λ	none
Z2	76	# At. no. 2nd heavy atom	none
FP_FPP2	-21.6 16.4	# f' f'' for Z2 at 2nd λ	none
	where λ stands for wavelength		
	(and optionally)		
RELWT_NATIVE	0.9	# relative weight of native	1
RELWT_DER1	0.5	# relative weight of 1st der.	1
RELWT_DER2	0.9	# relative weight of 2nd der.	1
RES_DER1	2.5	# intrinsic resolution of 1st der.	(native)
RES_DER2	1.9	# intrinsic resolution of 2nd der.	(native)
FSCALE_DER1	1.3	# fobs scaling factor, 1st der.	none
FSCALE_DER2	1.1	# fobs scaling factor, 2nd der.	none
AUTOSCALE	FALSE	# fine-tune fobs scaling?	TRUE

• Z[m]. The atomic number for the m-th derivative (MAD data). Nomally, if there are 2 sets of MAD data, $Z1 = Z2$.

• FP_FPP[m]. Values of f' and f'' for the m-th atom (MAD data).

• RELWT_NATIVE. This is the weight associated with the native data in the cost function. If you do not have an observed data set for the native, you should set *relwt_native* to 0 and use any available (dummy) file name for *fo_filename*.

• RELWT_DER[m]. This is the weight associated with the m-th derivative data relative to the native in the cost function. The default should be used in preliminary runs, but see also Section 5.3.2.

• FSCALE_DER[m]. This is the factor for scaling fobs data to absolute scale for the m-th derivative.

• RES_DER[m]. This is the resolution of the m-th data set, insofar as it differs from the native resolution.

• AUTOSCALE. It is our experience that the success of MIR or MAD runs is highly sensitive to the precise scaling of the fobs files among themselves. Generally, it is useful to fine-tune this scaling between outer iterations of the Solve process by enabling autoscaling. The code then changes the relative scales of the derivatives such that the overall cost function is minimized. The AUTOSCALE parameter allows you to by-pass this fine-tuning, if you wish.

A particular new procedure in setting up an MIR or MAD run, as compared with the single run discussed in Section 4.1 (with no MIR or MAD), lies in the scaling of the various data sets among themselves. The complete process is thus (Figure 5.1):

- problem definition
- resolution choice
- structure factor apodization (all hkl files)
- absolute and relative scaling
- consistent model preparation
- (optional) solvent target preparation

5.2.2 Preparation of Native Fcalc File

In a preliminary Solve MIR or MAD run, it is obviously best to start with some partial knowledge of the native electron density, based on the output of PHASES or MLPHARE, for example, suitably manipulated to convert it to X-PLOR/CNS format (see Section 3.2). Then the native fcalc file should be run through Apodfc. It must then be used to create a consistent model, as described for simple Solve runs, by running Back on it. If you have such a native electron density file, your Solve runs should be done in correction mode.

However, in your first Solve MIR or MAD run, the initial native structure factors may be unknown. In this case, use the special pseudo-name "empty" for the keywords FC_FILENAME and MD_FILENAME.

5.2.3 Preparation of Heavy Atom Fcalc Files

Fcalc files must be set up to contain structure factors corresponding to the heavy atoms, with appropriate occupancies. It is essential that the heavy atom structure factors be on an absolute scale. The program Phases, for example, does *not* scale the heavy atoms on an absolute scale. In the absence of any other program from the standard crystallographic repertoire, preparation of the heavy atom structure factors may be done using the Eden utility Tohu (see Chapter 11). It is possible to process MAD anomalous data in Tohu if you set keyword `ANOM` to TRUE. In that case, Tohu will calculate and write out structure factors for "hydrogen" atoms at the specified positions; further processing (using Z, f', and f'') is relegated to Solve. Note that although Tohu assumes point-like atoms, it does use the heavy atom occupancies and B factors and it does put the output on an absolute scale. In all cases, the resulting fcalc file should be run through Apodfc and inspected, to be sure that the least squares fitting of the plot to a linear approximation and the resulting smearing factor is appropriate. Finally, it must be put through Back to ensure a model gridded physical-space file for Solve. All of these preprocessors are discussed again in Chapter 8.

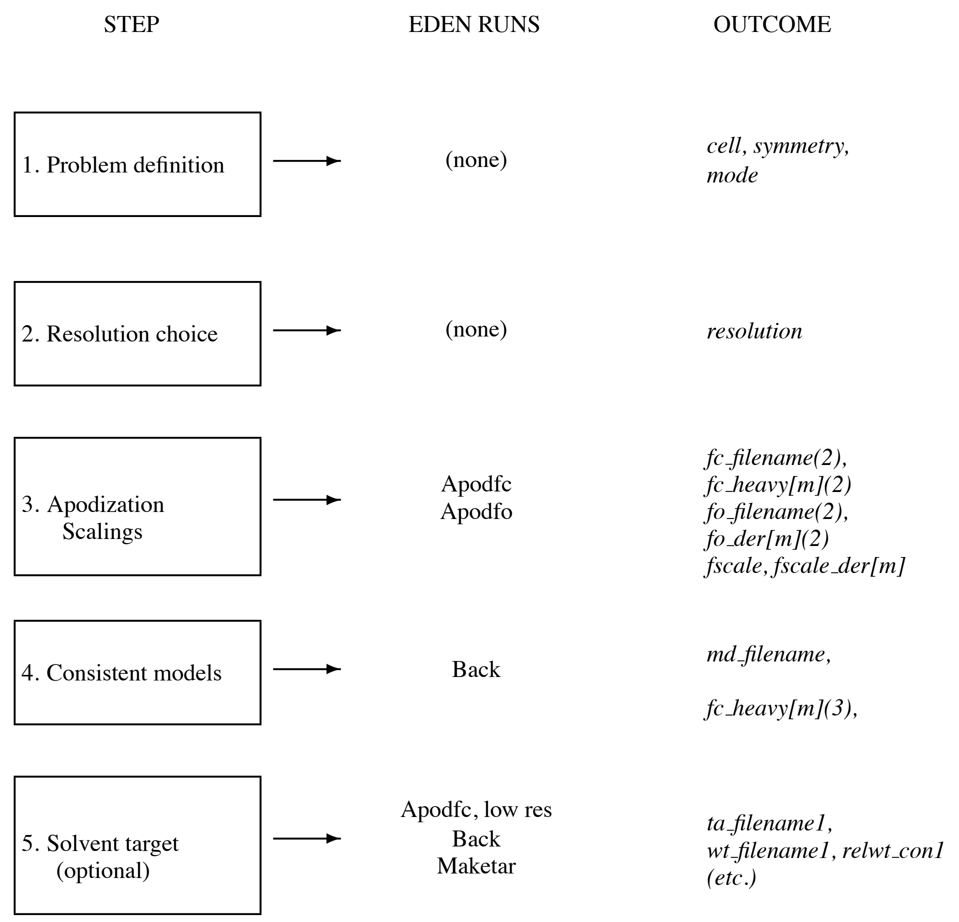

Figure 5.1: Preparations for Solve with MIR or MAD

5.2.4 Preparation of Derivative Fobs Files

Derivative fobs files may need to be apodized using Apodfo, exactly as the native fobs file was. Please remember that each file should contain an $F(0,0,0)$ term whose value is your best estimate of the total number of electrons in the unit cell of the MIR or MAD derivative data file, including both ordered and disordered solvent.

In principle, you can find the scale factor,*fscale_der[m]*, for scaling fobs to an absolute scale in the same way that you find *fscale* for the native. However, it is more accurate to find a relative scale factor *rel_fscale[m]* for the derivative with respect to the native. Then

$$fscale\_der[m] = rel\_fscale[m] * fscale$$

The value for $rel\_fscale[m]$ may be derived from the two intercepts reported for Apodfo when it is run using the native *fo_filename* and when it is run using the derivative *fo_der[m]*. If the reported intercepts are $y_{0,nat}$ and $y_{0,der}$ respectively, the required value is

$$rel\_fscale[m] = exp[(y_{0,der} - y_{0,nat})/2]$$

Since the intercepts are determined by extrapolating the linear approximation to the plots, they are also prone to error. Thus, although you may use this formula for estimating $rel\_fscale[m]$, it is safer and more accurate to work graphically. Let us assume that you wish to scale `Hgder.obs` to `nat.obs`. Running Apodfo on each of these files will give for each a Wilson plot; after you have chosen the δ_{fo} that is the best fit, Apodfo will write out files `nat.obs_wil` and `Hgder.obs_wil`. Now plot these two and look for a (positive or negative) constant, y_{diff} that, when added to `Hgder.obs_wil` will best bring it into coincidence with `nat.obs_wil`. Then use

$$fscale = exp[y_{diff}/2]$$

for scaling. In this way, the most reliable mid-resolution data can be used for scaling. If – as is usually the case – the derivative has a lower resolution than the native, you should apodize the native to the intrinsic resolution of the derivative (reported by Apodfo) for purposes of scaling.

5.3 Running Solve with MIR or MAD

5.3.1 Scaling Issues for MIR

Let N refer to the native and D to a typical derivative; for purposes of this discussion, we will consider the m-th derivative. Several parameters govern the behavior of the MIR solution process, the most important ones being: the scaling of fobs to fcalc data for the native, $fscale_N$; the comparable scaling of fobs to fcalc data for the derivative, $fscale\_der[m]$, which we refer to as $fscale_D$; and the occupancy of the heavy atom sites in the derivative, Occ_D. Another parameter in the cost function calculation is the weight associated with the cost function for the native, relative to the cost function for the derivative. When the fobs data have σ values, the relative weights of individual reflections are set automatically on their basis; otherwise, they are all set to be 1. Finally, the total number of electrons in the native and derivative molecules (including solvent) plays a role, but it is assumed that these numbers are well enough known.

Let ρ_N be the electron density of the native and ρ_H be the electron density of the heavy atoms in the derivative. The fundamental assumption is that the total electron density in the derivative, ρ_D, is

$$\rho_D = \rho_N + Occ_H * \rho_H$$

We shall write F(ρ_N) for $\|Fcalc_N\|$ and F($\rho_N + Occ_H * \rho_H$) for $\|Fcalc_D\|$. Then the conjugate gradient process seeks to minimize the sum of the squares of two terms — $term_N$ and $term_D$, where:

$$term_N = F(\rho_N) - fscale_N * Fobs_N$$

$$term_D = F(\rho_N + Occ_H * \rho_H) - fscale_D * Fobs_D$$

If the scalings $fscale_N$ and $fscale_D$ are correct, but the occupancy Occ_H of ρ_H is too low, the code will push some of the heavy atom contributions from ρ_H to ρ_N, decreasing $term_D$ and increasing $term_N$ in the process — i.e., the heavy atoms will "show through" in the resulting native electron density. Conversely, if the scalings are correct but the occupancy of ρ_H is too high, the code will cause the resulting native electron density to have holes at the positions of the heavy atoms. If the occupancies are correct but $fscale_D$ is too high with respect to $fscale_N$, the code will again add density corresponding to heavy atoms to ρ_N in the effort to minimize $term_D$. Again, this will cause $term_N$ to rise, but presumably the lowering of $term_D$ will more than compensate for it.

In summary, an MIR run that produces a native electron density with the heavy atoms showing through may be the consequence of either of two errors: too low occupancy of the heavy atoms in the derivative, or too high a scaling factor, $fscale_D$. Our experience has been that the latter error is more likely, since occupancies of 1 are common and occupancies cannot exceed 1.

5.3.2 Preliminary Runs

Based on the ideas that were just discussed, we recommend that you start off the MIR runs at a relatively low resolution (say, 3 Ångstom), without a solvent target. This will allow you to do a number of runs fairly rapidly. In order to decide whether the occupancies and scaling are well tuned, you will need a display program that allows you to examine two-dimensional slices of the binary data produced by Solve. We are delighted to inform our readers that the latest version of PyMOL (experimental version 0.98) is capable of letting you examine slices in a *quantitative* fashion and, for example, enabling you to pinpoint unusually high or low densities at well-defined positions.

For Solve runs in general, we recommend that you use the `-v` option; this will produce a listing of the cost function components. A careful study of the sizes of the native and derivative cost functions will indicate the degree to which the derivative data sets are reliable. If you find that a certain derivative seems to give consistently high cost function values, compared to the native and other derivatives, you may want to suppress its contribution by selecting a low value for $relwt\_der[m]$ or even by omitting the derivative entirely.
Once the data are pronounced well-scaled and the occupancies are correct, you are ready for a high-resolution run. The only change in the preparation steps is that the starting model, that was previously either an empty file or the output of Mlphare, for example, may now be replaced by the best result from your preliminary runs. Be sure that *all* the preparation steps, including all the apodization runs, are redone at the new *resolution*.

5.3.3 Isomorphism Checks

The result of the high-resolution MIR run may be regarded as an approximation to the native crystal, possibly warped in the region of the heavy atoms where the native and derivatives did not display complete isomorphism. Thus, you may wish to explore the native and derivative crystal structures separately. This section deals with our (limited) experience in this regard. See also Section 6.2.

Consider an MIR run identified by its input file, say `abc.inp`. We would like to remove the bias in its outputs that arose as a result of the "not-so-isomorphous" replacement. We take the native output file of the MIR run, `abc.bin` and use it as input to a second, ordinary (non-MIR) Solve run. However, in order to make sure that the Solve solution does not stray too far from our previous results, we *also* use the binary file as a target! (The accompanying WT_FILENAME should have the value `full`.) We have found that a *very* low value of the relative weight such as $3 * 10^{-3}$, is sufficient to keep Eden from straying too far from its starting point, while still allowing the solver to correct errors from the original MIR run.

We then do similar runs for the derivatives, but now, we will need to do a couple of extra steps. First, we prepare a structure factor file corresponding to the native result of the MIR/MAD run, by running Forth on `abc.bin`. Then add to it the original structure factor file for a selected (m-th) derivative. For this you will need a little utility, Cadhkl described in Chapter 11.3. Then, run Back to prepare a consistent physical-space model. Now, from this starting point, run "ordinary" Solve as described above for the native. If you subtract `abc.bin` from the output of this new Solve run, you will find the correct positions and shapes of the heavy atoms, as well as the distortions of the protein caused by those heavy atoms. You can run Forth on this difference and use the result as an improved fc_heavy[m]. A 2nd full MIR run using all the (hopefully) improved starting points shold give a better-defined electron density for the native.

Chapter 6

Physical Space Constraints

6.1 Overview

As described in Section 4.2, the search for an optimal set of electrons per voxel is conducted in physical space. The cost function value has both Fourier-space components, of which the difference between observed and calculated structure factors is the principal component, and physical-space components. It is the sum total of all the components of the cost function that is used to guide the conjugate gradient solver in its search for a minimum. At this point, we will discuss the physical-space components.
There are various kinds of physical-space (N_p) constraints[1] in Eden's Solve program that may be applied, together with the N_{hkl} space constraints, at each inner iteration of the optimization process. One of these, target constraints, may be considered quite general in application. All the others are more specialized; they may be appropriate only within a limited resolution range, for example. A comprehensive discussion of spatial constraints is to be found in [16].

By default, there are no physical-space constraints: $nconstraints = 0$. The value of NCONSTRAINTS is limited to a maximum of 12. In fact, it seems unlikely that more than 2 – 3 would be useful when applied simultaneously. All constraints require two input keyword-value pairs — CON_TYPE[n], which identifies the kind of constraint, and RELWT_CON[n], which identifies the relative weight to be associated with that constraint, where [n] stands for a number of range $(1, nconstraints)$. There are other inputs that are specific to the constraint type; they will be introduced individually in the following sections of this chapter. Legal values for $con\_type[n]$ are: [2]

`target` for a solvent or protein target.

`solvent_tar` for a solvent target.

`stabilize_tar` for a protein target.

`phase_ext` for Eden's version of phase extension,

[1]The term *restraints* would, in fact, be more suitable for our cost functions, since they do not absolutely constrain the solver but instead, "encourage" it to a greater or lesser degree.

[2]The distinctions among `target`, `solvent_tar` and `stabilize_tar` are for reporting purposes only; the Solve code does not actually distinguish one from another.

`cs` for crystal symmetry.

Values of relative weights are typically in range $10^{-3} - 1$. See also Section 6.5.

Each type of physical-space constraint will now be discussed.

6.2 Targets

Target constraints require the kind of input described in Table 6.1 — i.e., in addition to the standard input (for all physical-space constraints), they require the names of two sets of files in physical space. One, $ta\_filename[n]$, contains the electron/voxel values that will be used as targets; the other, $wt\_filename[n]$, contains the weights associated with these values. Weights may be in range (0,1), but generally they are either 0 or 1. There is a special pseudo-name — "full" — that may be used with keyword WT_FILENAME, signifying that all electron/voxel values are to be given a weight of 1.

Table 6.1: Target Constraint Input for Solve

Keyword	Example of value	description	default
NCONSTRAINTS	1	# no. of target constraints	0
CON_TYPE1	target	# description of 1st constraint	none
RELWT_CON1	0.1	# rel. wt. for 1st constraint	0
TA_FILENAME1	mytarget	# name of 1st Np space target	none
WT_FILENAME1	myweight	# name of 1st Np space weight	none

Target constraints are applied in Eden's Solve in the following form ([16]):

$$f_{target} = Relwt * Const * \sum_{p=1}^{N_p} wt_p^2 (n_p - n_{p,targ})^2$$

where n_p is the current electron/voxel value at a point p, $n_{p,targ}$ is the targetted electron/voxel value at that point, and wt_p is the weight associated with the target at that point. Note that wherever the weight is zero, the density is *not* constrained – i.e., it is free to change. The value of Const is proportional to the fraction of total points that are targetted by the weight file.

Currently, target constraints may be applied in three scenarios: (a) for enforcing a well-established partial model; (b) for a stabilizing target; and (c) for a solvent target.

(a) If you have a good partial model, the weight array should cover the partial model alone (i.e., it should differ from 0 only where the current model is significantly greater than 0). A high relative weight ($0.1 - 1.$) is appropriate. Note that a partial model target is potentially a stronger constraint than the basic "completion" mode of operation of Solve: when the partial model is regarded as a *target*, its value will be maintained more or less unchanged — electrons will be neither added to it nor subtracted from it — but when Solve operates in completion mode without a target, electrons may be freely added to the partial model.

(b) A stabilizing target is similar to a completion mode target, except that the model should be essentially complete. A weight array containing all 1's and a low relative weight are appropriate. A stabilizing target is useful in almost any run; its purpose is to keep Eden from straying unnecessarily from the starting model. Note that if there is too much freedom, i.e., not enough data, Eden is free to put "garbage" anywhere – and it will.A stabilizing target ensures that the phase changes introduced by Eden are the smallest compatible with the information supplied (such as the diffraction pattern, positivity, derivative information and solvent regions). Insofar as there is no such information, Eden recovers a difference Fourier map.

(c) Solvent targets are probably the commonest form of constraint used in Eden. Our experience is that solvent targets are especially helpful for extending the scope of Eden's power to solve protein structures (see [14]).

Eden's Maketar was designed for preparing the weight files needed for all target constraints. See Section 8.4. For a protein target, you should run Maketar, setting TARGET to "high"; in this way, the targetted points will cover the protein rather than the solvent area. The $mask\_fraction$ or $threshold$ input allows you to fine-tune the fractional level at which weighting kicks in. For a solvent target, the target array should contain a value of about 0.34 el/cubic Ångstrom (converted to units of electrons/voxel) and the weight array should cover whatever region is established to be the solvent, using X-PLOR/CNS, for example, followed by Eden's Back. Watch out with this procedure! X-PLOR/CNS prepares the solvent region with a positive value, the non-solvent region with value 0, so from the viewpoint of Maketar, this is a *"high"* target.

An alternate (safer) way to prepare a solvent target is to run Apodfc with a very low resolution (high value of input APOD_RES) to prepare a smeared-out version of the known model; then run Back at the regular resolution; and finally, run Maketar setting TARGET to "low", thus targetting the solvent. You may have to experiment to find a relative weight that is large enough to be effective, but not so large that the edges of the solvent region are clearly visible in a false-color rendition of the final output. For determining the relative weight, you should examine the cost function report. Typically, at least in the first outer iteration, the target contribution should be relatively small; later, it should approach the hkl contribution or even surpass it, but not by orders of magnitude.

6.3 Phase extension

The input for a phase extension constraint includes the same information as for target constraints, as well as a phase extension resolution. See Table 6.2. Let us imagine that a credible .fcalc model of the problem at a resolution of 6 Ångstrom has been established. Call it prot6.hkl. We may use this solution as the FC_FILENAME and extend our knowledge of the protein details to higher resolution — say, 2.5 Ångstrom

— in the following manner. We run Back using resolution = 2.5, to obtain a real-space counterpart for prot6.hkl, but at a grid spacing that is compatible with the intended (higher-resolution) run. This model (call it prot6m) will serve *both* as MD_FILENAME *and* as TA_FILENAME. This is a case where the appropriate WT_FILENAME may be "full" (i.e., all points will be assigned a weight of 1.0, without any need for preparing a special bin file).
We will thus have the special input for phase extension as shown in Table 6.2.

Table 6.2: Phase Extension Constraint Input for Solve

Keyword	Example of value	description	default
NCONSTRAINTS	1	# Number of constraints	0
CON_TYPE1	phase_ext	# Description of constraint	none
RELWT_CON1	1.e-4	# relative weight	0
PHASE_EXT_RES	6	# inherent resolution of target	none
TA_FILENAME1	prot6m	# target file name	none
WT_FILENAME1	full	# weight (pseudo) file name	none

The application of phase extension uses a cost function that is applied in reciprocal space (see [16]). Note that a phase extension constraint should always be applied in correction mode. Note too that if your low-resolution model is only partial, you can make a real weight file, as discussed previously.

6.4 Crystal Symmetry

There is no special input for imposing the crystal symmetry constraint at each step in the optimization process, other than specification of CON_TYPE[c] and RELWT_CON[c], where [c] stands for the index of the crystal symmetry constraint. Our experience is that there is little to be gained from application of this cost term.

6.5 Choice of Relative Weights

The value of the relative weight represents the weight of the n-th physical-space cost function relative to the (hkl) space cost function. As stated previously, the proper value of the relative weight can be anywhere in range $10^{-3} - 1$. In the absence of MIR or MAD, the (hkl) space relative weight is 1. Otherwise, it will be greater by approximately the number of derivatives. In order to get a handle on the useful relative weight for a typical physical-space constraint, you should run Solve with the `-v` option [3] and examine the `.cost` file. Assume a single N_{hkl} cost and a single N_p space cost. If from the start, the physical-space cost outweighs or is comparable with the N_{hkl} space cost, the relative weight is too large. Our experience is that in the

[3] In fact, we recommend *always* running Solve with the `-v` option

first outer iteration, the N_{hkl} term should dominate, while in the 2nd outer iteration (where Solve generally works hardest), the two spaces should contribute comparable amounts.

Chapter 7

Reciprocal Space Constraints

7.1 Overview

There are currently two forms of reciprocal-space constraints in Eden. One is for singlet and triplet invariants, which may be incorporated in the cost function calculations most effectively. The other is a detwinning algorithm which may be applied in two modes – either amplitude or intensity detwinning (with input fraction). Currently, these constraints are applicable for a native data set only – i.e., they may not be combined with MAD or MIR.

7.2 Singlet and Triplet Constraints

Knowledge about singlet and triplet phases is obviously useful to Eden. The singlet reflections are the origin-determining reflections and the semi-invariants consistent with them. The triplets are triplet invariants. The knowledge is not applied blindly, since it generally comes with some error allowance. Instead, at each point in the generation of consistent phases, the singlet and/or triplet invariants is used with its error allowance and with an input weighting coefficient to give two new costs, which are added into the general cost function.

The singlet invariants come in a file (identified by name *sfile* in Table 7.1, with no added suffix) and is expected to contain 5 space-delimited numerical fields:

$$h\ k\ l\ phase\ sigma$$

The triplet invariants come in a file (identified by name *tfile* in Table 7.1, with no added suffix) and is expected to contain 11 space-delimited numerical fields:

$$h1\ k1\ l1\ h2\ k2\ l2\ h3\ k3\ l3\ phase\ sigma$$

where

$$\sum_{i=1}^{3} h_i = \sum_{i=1}^{3} k_i = \sum_{i=1}^{3} l_i = 0$$

Eden uses this information as a soft restraint at each step of the cost function minimization algorithm.

Table 7.1: Singlet and Triplet Invariant Input for Solve

Keyword	Example of value	description	default
NCONSTRAINTS	2	# Number of constraints	0
CON_TYPE1	singlet	# Description of constraint	none
RELWT_CON1	1000.	# Singlet relative weight	0
TA_FILENAME1	sfile	# Singlet target file name	none
CON_TYPE2	triplet	# Description of constraint	none
RELWT_CON2	30.	# Triplet relative weight	0
TA_FILENAME2	tfile	# Triplet target file name	none

7.3 Detwinning

When a crystal is merohedrally twinned, there are other programs that can deal with intensity twinning which may or may not be entirely successful. The possibility of amplitude twinning is not handled in any conventional crystallographic package, to the best of our knowledge. Therefore, we have introduced a detwinning package to deal with both sorts of twinned crystal. This should be looked upon as a reciprocal-space constraint, even though the input follows a different scheme from the usual pattern (target filename, relative weight, etc.) See Table 7.2.

Table 7.2: Detwinning Input for Solve

Keyword	Example of value	description	default
DETWIN	TRUE	switch to activate detwinning	FALSE
T_TYPE	I	'I' (intensity) or 'A' (amplitude)	none
T_MATRIX	1 0 0 -1 -1 0 0 0 -1	twinning matrix	none
T_FRAC	0.2	twinning fraction (range: 0 - 0.5)	none

• DETWIN. A switch that you may set to TRUE if you believe that the data are twinned to some degree. Only if DETWIN is TRUE will the following 3 keywords be read.

• T_TYPE. A character (A or I) indicating whether the twinning was amplitude or intensity; Eden has different algorithms for the two cases.

• T_MATRIX. The 3-by-3 matrix transforming the reflection index to the index of its twin.

• T_FRAC. The fractional extent of twinning.

Chapter 8

Preprocessing Utilities

Up to this point in the manual, there have been many references to the preprocessing utilities that are needed for setting up Solve runs. We now discuss each preprocessor in detail.

8.1 Apodfc and Apodfo

Apodization is the technical name for changing the crystallographic B-factor in order to smear out the electron density. This is required so that it may be represented with fidelity in terms of Gaussian blobs on a grid. The two apodization programs, Apodfo and Apodfc, carry out an analysis of the structure factor data that is similar to a Wilson plot. They are used for determining the scale factor that places the fobs on an absolute scale ($fscale$), as well as smearing factors for the fobs and fcalc (δ_{fo} and δ_{fc}). The smearing factors are used to adjust the resolution of your data to the intrinsic resolution of the Eden solver.

Apodfo reads structure factors from an input fobs file, while Apodfc reads structure factors from an input fcalc file. *Please note that the fobs information should be entered in terms of amplitudes and amplitude sigmas,* **NOT** *intensities and intensity sigmas!* Each utility generates a set of data points that are mean values of $\ln(\|F\|^2)$ within shells ("bins") of $1/d^2$, where F stands for F$_{obs}$ or F$_{calc}$ and

$$1/d^2 = (h^2/a^2) + (k^2/b^2) + (l^2/c^2)$$

or its general form for non-orthogonal crystals [6]:

$$1/d^2 = ((1 - \cos^2\alpha)(h^2/a^2) + (1 - \cos^2\beta)(k^2/b^2) + (1 - \cos^2\gamma)(l^2/c^2)+$$
$$2(cos\beta cos\gamma - cos\alpha)(kl/bc) + 2(cos\gamma cos\alpha - cos\beta)(lh/ca) + 2(cos\alpha cos\beta - cos\gamma)(hk/ab))/$$
$$(1 - cos^2\alpha - cos^2\beta - cos^2\gamma + 2cos\alpha cos\beta cos\gamma)$$

Given an input resolution, each utility then finds the slope of that set of data points, using appropriate resolution limits and uncertainties (see below). The slope is equivalent to a global crystallographic B factor. Each one reports that B factor and the y-intercept ($y_{0,obs}$ or $y_{0,calc}$) of the linearly-fit data, to be used for scaling the experimental data. Insofar as the smearing factor is greater than zero, the apodized version of the input structure factors is written out. In fact, there are 2 resolutions that participate in the apodization: $resolution$ – the usual variable – is used for either accepting or discarding input structure factors; $apod\_res$ is a variable unique to these utilities; it determines how strongly the program will apodize. By default, $apod\_res = resolution$, but you may choose a larger value if you wish to smear the information more strongly (e.g., for preparing a solvent target).

Apodfc and Apodfo then find the slope of that (very roughly) linear function. They use two methods for deriving the slope: one is a straightforward least-squares minimization; the other more sophisticated method uses a "universal" correction factor [3] that suppresses much of the non-linearity specific for globular proteins. The non-linearity comes from their solvent content that decreases the density contrast at low resolution. If you run Apodfc and Apodfo with the `-g` flag, graphs using both methods are presented for your inspection (under Xmgr) and we also print out our recommendation in the terminal report – but you may make your own choice. If you run them without the `-g` flag, Eden decides which method to use, based on minimizing the standard deviation of the linear data with respect to the original data.
Run Apodfc by typing:

`eden [-gv] apodfc` *name sfname*

where *name*`.inp` is the input parameter file without extension `.inp`, *sfname* is a structure factor file name typed in its entirety,

optional `-g` (graphics) invokes `xmgrace` and displays plots of the mean values of $\ln(\|F\|^2)$ as a function of $1/d^2$. There are 4 such plots — the original binned data; the best linear fit to that original data; data corrected using a universal correction for protein non-linearity; and the linear best fit through the corrected data. The use of the `-g` option is highly recommended. However, if you do not have `xmgrace` on your system (and thus do not invoke this option), the files that are used for the simple $x - y$ plots will be written out and are thus available for you to inspect with some other plotting program.

optional `-v` (verbose) produces a number of extra files that are unlikely to be of interest to the casual user.
Similarly, run Apodfo by typing:

`eden [-gv] apodfo` *name sfname*

where *name*`.inp` is the input parameter file without extension `.inp`, *sfname* is an fobs file name typed in its entirety,
Apodfc and Apodfo both expect to find an input parameter file, *name*`.inp`, containing run conditions and parameters, entered as upper- or lower-case keywords (first column) followed by values (second column). Use a "generic" input file (see Table 3.1) plus optional information from Table 8.1.
Usually, there is no need to use non-default values for BINWIDTH, MAX_RES or MIN_RES — but see below.

Table 8.1: Optional Input for Apodfc and Apodfo

Keyword	Example of value	description	default
BINWIDTH	0.004	# width of intensities shells	0.002 1/Å^2
MIN_RES	4.0	# minimum resolution	3.5 Å
MAX_RES	1.9	# maximum resolution	0.05 Å
APOD_RES	6.0	# apodization resolution	$resolution$
	(and for Apodfo only)		
USESIG	FALSE	# switch to use fobs SIGMA	TRUE

The weighted linear fit is calculated over a subset of $1/d^2$ space, corresponding to the available extent of (hkl) in the input (fcalc or fobs) file and limited by the range (min_res, max_res). Weighting is determined by the number of reflections in each bin and (for fobs apodization) by their sigma values. If the -g flag is in effect, the mean values of $\ln(||F||^2)$ within each shell vs. $1/d^2$ and a linear fit to those values are written to text files `wil` and `lin_wil`, respectively, for inspection with the plotting program `xmgrace`. Adjusted versions of the two files that correct for the universal shape are also written and displayed as `wil_w0corr` and `lin_wil_w0corr`. We recommend that you study the plots to be sure that the fit is good. If not, for example if the linearized plot extends to too low values of $1/d^2$, you may want to enter an adjusted (lower) value for keyword MIN_RES.
After you have chosen the smearing factor, the selected Wilson plot will be written to *sfname*`_wil`. (While MAX_RES is also available for changing the upper limit of $1/d^2$, we have seldom found a need to fiddle with it.) Please note: changes in MIN_RES or MAX_RES have to do with the limits on the x-axis over which linearization will be applied. Do *not* change INPUT_RES to be "consistent" with them! — INPUT_RES affects the calculation of δ_{fo} or δ_{fc} critically. If either apodization utility reports an error: "Trouble! - empty bin(s) ...", followed by a list of bin occupancies and values of $||F||^2$, you should increase the value of BINWIDTH judiciously from the nominal value of 0.002 to $n * 0.002$, where n is an integer.

Normally, the codes will write apodized structure factors to a file whose name is derived from the input structure factor file, by adding `_apo` before the file extension. However, sometimes Apodfc or Apodfo will report a negative smearing factor. That means that your data has a lower intrinsic resolution (higher B value) than the solver can provide. This is not a problem; Apodfo and Apodfc will not write out apodized files. You should use the input ("unapodized") files for all further processing.

8.2 Expandfc and Expandfo

Expandfc and Expandfo expand structure factor files to $P1$. Solve and Back now quietly expand data to $P1$ (which was not the case in earlier versions of Eden). Nevertheless, Expandfc and Expandfo runs may occasionally have to be a part of the preprocessing of .fcalc and .fobs file in your problem. The reason for this is that Eden works in the upper half-ellipsoid ($h \geq 0$), which is not necessarily the case for the programs that produced your files.

Consider first Expandfc; run it by typing

`eden expandfc` *name fc_filename.ext*

where *name* stands for the parameter file name without extension `.inp`, containing run conditions and atomic parameters, entered as upper- or lower-case keywords followed by values (see Table 3.1). There is generally no special input for Expandfc. However, use the keyword/value pair `ANOM TRUE` for anomalous dispersion files, for which Friedel's relation does not hold. Otherwise, the utility will report very large numbers of mismatches, which it finds when trying to satisfy that relation and you will lose the anomalous information.

Similarly, run Expandfo by typing

`eden expandfo` *name fo_filename.ext*

where *name* stands for the parameter file name as before (see Table 3.1). You should use the keyword/value pair `ANOM TRUE` for anomalous dispersion files, for which Friedel's relation does not hold.

Although Expandfc and Expandfo require input of only the unique set of reflections, they read all reflections, expand them, and verify that the expansions are consistent. It sometimes happens that expansion of the original model does not produce consistent values. For example, we have observed data generated by Phases from a hexagonal crystal for which the centric reflections at $n * 60°, n \neq 3$, were off by as much as a degree. In that case, Eden will report "mismatches", the first 20 of which will be written to the log. If you run the Expand utilities with the verbose switch, all the mismatches will be written to the log. Do check these to be sure that there isn't some real error in the crystal classification. Regarding the naming of the output of these programs, consider running Expandfc on *fc_filename.ext*; the output file will be named *fc_filename*`_P1.`*ext* for ordinary data; *fc_filename*`_P1plus.`*ext* for anomalous data from a crystal that is not triclinic; and two files named *fc_filename*`_P1plus.`*ext* and *fc_filename*`_P1minus.`*ext* for anomalous data from a triclinic crystal.[1] Corresponding names apply to the output of Expandfo.

The expansion preprocessors in Eden do a simple expansion of the data in your input files to $P1$. Whenever the expression "expanded to P1" appears in this manual, the meaning is the unique points in the $h \geq 0$ half-ellipsoid in (h, k, l) space:

[1] In fact, *fc_filename*`_P1minus` has the complex conjugate of *fc_filename* for the negative *h* hemi-ellipsoid. Eden Solve uses those as such.

$$0 < h \leq \infty, \ -\infty \leq k \leq \infty, \ -\infty \leq l \leq \infty,$$

$$h = 0, \ 0 < k \leq \infty, \ -\infty \leq l \leq \infty,$$

$$h = 0, \ k = 0, \ 0 \leq l \leq \infty$$

.

Normally, you will *not* need to expand your fobs and fcalc explicitly; the expansion will be done internally, in Back and Solve.

8.3 Back

Back estimates electron/voxel data from a set of calculated structure factors, such as a starting phase set. It obtains a "solution map": the amplitudes of a set of Gaussian densities of given width, $\sqrt{\eta} * grid\_spacing$, centered on a simple grid or on a body-centered grid of given grid spacing. The code reads the diffraction pattern to the appropriate resolution and sets up a physical-space map on a grid at resolution *grid_spacing*, where $grid\_spacing = 0.6 * resolution$ for a simple grid and $grid\_spacing = 0.7 * resolution$ for a body-centered grid. It varies the number of elements in each grid cell in this map, such that their fast Fourier transform, multiplied by $F(hkl)$ by $exp[-\eta * \pi^2 * (dr)^2 * |h|^2]$ is optimally close to the input diffraction pattern. Note that Back is *not* simply a back-FFT of the starting phase set, Such a procedure could produce negative electron/voxel values, that Eden abhors. Rather, Back (like Solve) applies a conjugate gradient optimization search to find the set of non-negative electron/voxel values that are the best fit to the input phase set.

The main purpose of this calculation is to provide a "known" map to serve as its input model to Solve. Another purpose is to prepare a higly smeared map that will serve as the basis for preparing solvent targets for Solve. When the solver is run in "completion" mode, Back's output values will provide initial lower bounds on the solver .

Run Back by typing

```
        eden [-v] back name
```

or

```
        eden [-v] back name fc_filename
```

where *name* stands for the parameter file name without extension `.inp`, containing run conditions and atomic parameters, entered as upper- or lower-case keywords followed by values. See Table 8.2 (which is a subset of the input described in Table 3.2). Using the first form, you should enter the name of the fcalc file as an entry in the input file (FC_FILENAME), Using the second form, that name appears on the execute line (*fc_filename*). Use of the verbose switch (`-v`) causes Back to write the value of the cost function at each iteration into a file named *name*`.cost`. This is seldom of interest!

Table 8.2: Input for Back

Keyword	Example of value	description	default
	basic input for all Eden programs (see Table 3.1)		
SYMMETRY	P3221	# space group name	none
CELL	57.2 33.9 68.7 90 90 120	# unit cell dims in Ångstrom	none
		# and angles in degrees	none
RESOLUTION	2.0	# resolution in Ångstrom	none
RECORD	myrecord	# file name for a brief report	history
	other required input for Back (if not on the execute line)		
FC_FILENAME	k.fcalc	# calculated struc factor file	none
	uncommonly used input for Back		
DFDX_CRIT	0.003	# decrease in gradient to end run	0.001
R_STOP	0.03	# R factor to end run	0

Back uses an optimization process that is very similar to that of Solve (see Section 4.2) but without the outer iteration loop. Like Solve, Back writes a full log and (if the verbose switch is invoked) a listing of the cost function values. Back no longer writes out a new structure factor file that is consistent with the electron/voxel file, since such a file is no longer used as input to Solve. If you need such a structure factor file, you should run Forth on the electron per voxel file that Back writes. The electron/voxel file that Back produces may be regridded, using the postprocessor Regrid, just like Solve output electron/voxel files. The regridded *map* file may then be used to view the starting model.

8.4 Maketar

The principal purpose of Maketar is to prepare solvent targets and weights for Solve runs. Another use is to prepare stabilizing targets for a partial model. Run Maketar by typing:

`eden maketar` *name modfile*

where *name* is an input parameter file name without extension `.inp` and the electrons/voxel will be taken from *modfile*`.bin`. For a solvent target, the electrons/voxel file is typically prepared by running Apodfc at a very low resolution (e.g., 7 Ångstrom). The Apodfc output then serves as the FC_FILENAME in a Back run at the regular resolution, providing electron/voxel files for use by Maketar. Points in the input electron/voxel files are redefined as "low" and "high" such that the (input) fraction $mask\_fraction$ are targetted. You may replace a global $mask\_fraction$ by another input, $threshold$, whose value in electrons/cubic Ångstrom (after suitable conversion to electrons/voxel) designates the level below which voxel values are "low".

The sense of the weight file is determined by the obligatory input, $target$ whose possible values are “low” or “high”. If “low”, the low voxel values have weights of 1; if “high”, the high voxel values have weights of 1.

Maketar expects to find an input parameter file, *name*`.inp`, containing the usual basic parameters plus some special input parameters. See Table 8.3. Output is written to binary files named `weight` and `target` with the usual `.bin` extension. The weight files contain 1’s at targetted points, 0’s elsewhere. The target files, which are useful only for solvent targets, contain $target\_value$ at all points. In fact, the contents of the target files at points for which the weight is 0 are irrelevant.

Table 8.3: Input for Maketar

Keyword	Example of value	description	default
	basic input for all Eden programs (see Table 3.1)		
SYMMETRY	P3221	# space group name	none
CELL	57.2 33.9 68.7 90 90 120	# unit cell dims in Ångstrom	none
		# and angles in degrees	none
RESOLUTION	2.0	# resolution in Ångstrom	none
	other obligatory input for Maketar		
TARGET	low	# “high” or “low”	none
MASK_FRACTION	0.4	# fraction of points targetted	0.5
	... or		
THRESHOLD	0.30	# threshold density, in el/Å^3	
		# for targetting	none
	optional input for Maketar		
RECORD	myrecord	# file name for a brief report	history
TARGET_VALUE	0.25	# cutoff in el/Å^3 for target	0.34

• TARGET. This is an obligatory input whose value is “low” or “high”. Use “low” for a solvent target prepared with Apodfc; use “high” for a solvent target prepared with X-PLOR/CNS or for a stabilizing (protein) target. If $target$ is low, the low points, as determined by the $mask\_fraction$ or $threshold$, will have their weights set to 1. If $target$ is high, the high points, as determined by the $mask\_fraction$ or $threshold$, will have their weights set to 1.
• MASK_FRACTION. This specifies the fraction of all points in the electron/voxel input file that should be targetted and hence defines the level separating low from high points.
• THRESHOLD. This is an alternate way of defining the level separating low from high points. It sets the

limiting low value in terms of el/Å^3.
• TARGET_VALUE. This specifies the electron density that will be written into all positions in the "target" file. For a solvent target, typically you will use the default value of 0.34 electrons/Å^3. For a stabilizing target, this input is superfluous — in fact, the file named "target.bin" may be discarded. Instead, the input file (*modfile*) will serve as the target.

Please note that if you change the resolution at which Eden is working, you must rerun both Back (to prepare the physical-space model) and Maketar.

8.5 Sym

Sym is a utility for manipulating pdb information directly. It is used in Eden for two purposes: (a) reporting points of crystallographic symmetry in the unit cell; (b) identifying fractional limits in the /pdb file for Regrid (see Chapter 9).
Run it by typing

```
eden [-i] sym sname pdbname
```

where *sname* stands for the input parameter file name typed without its `.inp` extension and *pdbname* stands for the pdb file name with or without its `.pdb` extension.

optional `-i` stands for interactive mode; you will be prompted to enter atomic coordinates from the terminal and Sym will report to you all points related to your input by crystallographic symmetry. In interactive mode, *pdbname* is not needed.

no `-i` stands for non-interactive (default) mode. In this case, Sym reports the extent of the pdb information after expansion, in terms of fractional values along the crystallographic axes.

Sym expects to find an input parameter file containing the information described in Table 3.1 Optionally, you may use keyword OVERLAP with a value *dist* (only in non-interactive mode) for checking purposes; it tells Sym to report and eliminate any atom in the pdb file if it overlaps another atom (in another asymmetric unit within a distance of *dist* Å. Such atoms are eliminated from all equivalent positions.

Please do check the output of Sym with regard to the atoms found; Eden expects that the ATOM (or HET-ATM) information is written according to the CCP4 specifications, but we have sometimes encountered 'illegal' cases. Pdb files are read by Sym, Tohu and Count utilities
The Sym and Tohu logs include Matthews' coefficient [2], which is defined as vol/mass of the full cell. They also include the protein volume fraction.

[2] 'The CCP4 Suite', p.49 - 50.

Chapter 9

Viewing Eden's Results

9.1 Regrid

Regrid takes as input a set of physical-space files in electrons/voxel that are the result of a Solve (or a Back) run. It produces an electron density map in units of electrons/cubic Ångstrom on a grid that is *N* times finer than the input, where *N* is a small integer (by default, 2). For the default η, a 2:1 regridded map produces data on a grid that is $\approx$ 3 times finer than $resolution$. This is the usual practice in crystallography. For a body-centered grid type, the value of *N* must be even. For non-default values, *N* is read from the execute line:

`eden regrid` *name sname [N]*

Regrid expects to find an input parameter file *name*`.inp` and, the binary file *sname*`.bin`. Regrid assembles a single electron density map from the binary information. It writes an X-PLOR/CNS file *sname_N*`.map` in the standard format, ready for viewing in O (after running Mapman [9]). If you use PyMOL, switch the extension `.map` to `.xplor`.

If you display electron densities with XtalView in place of O you should follow an Eden Solve by running Forth, and then an awk script, `awk_xplor_to_xtal`, to be found in Appendix B. You should then skip the Regrid postprocessing entirely.

Regrid uses the usual input parameter file containing run conditions and parameters plus some unique input. See Table 9.1. The values for X_LIMITS, Y_LIMITS and Z_LIMITS may extend over negative or positive fractional ranges, depending on the region of visual interest. If you don't know what the ranges should be but you have a fairly complete pdb file, Regrid can use it to derive the appropriate ranges.

Table 9.1: Input for Regrid

Keyword	Example of value	description	default
SYMMETRY	P3221	# space group name	none
CELL	57.2 33.9 68.7 90 90 120	# unit cell dimensions in Å	none
		# and angles in degrees	none
RECORD	myrecord	# file name for a brief report	history
RESOLUTION	2.0	# resolution in Ångstrom	none
	commonly-used optional input for Regrid		
X_LIMITS	-0.5 0.5	# x limits in fract coords	0 1
Y_LIMITS	-0.6 0.4	# y limits in fract coords	0 1
Z_LIMITS	0 1	# z limits in fract coords	0 1
	or		
PDB_FILENAME	mypdb	# for deriving [XYZ]_LIMITS	none
	rarely-used optional input for Regrid		
HIGHRES	TRUE	# special high-res processing?	FALSE

Chapter 10

Evaluation Utilities

10.1 Overview

In addition to viewing the electron density with O, XtalView or PyMOL, Eden provides some tools for quantitative evaluation of its results. These tools are complementary to the usual tools for refinement – i.e.fitting models.

10.2 Count

Count counts the electrons in the environment of each atom in an associated pdb file, using as its source the result of a Solve run. Count assumes that atoms are spherical and have a Gaussian fall-off in space; it deals correctly with partitioning electron density among overlapping atoms. This is a useful utility for gauging the success of a high-resolution Solve run.

The invocation for Count is:

`eden count` *name sname [N]*

where *name* stands for an input parameter file name *sname* stands for the base name of the binary file to be counted, and *N* stands for a regrid factor (2 by default)

Internally, Count applies the Regrid algorithm before counting. Thus, input may also include the "regrid factor" used in Regrid; however, a value other than the default (2) is unlikely to be useful.

There are 3 special keywords for Count — see Table 10.1: the mandatory PDB_FILENAME and BCORR, and optional LEVELS. PDB_FILENAME is self-explanatory. BCORR is a correction to the pdb file B values, that is normally 0 but may be changed if the fobs file was apodized prior to running Solve. In this case, use the value reported in the `apodfo.log` file.

LEVELS sets the fractions of RESOLUTION defining radii for counting. They are the multipliers of radii corresponding to the (corrected) B values of the atoms.

Count writes an ASCII file, *sname*`_N.count`, containing most of the pdb file information plus the electron count around each atom, extended out to 1, 1.5 and 2 radii, by default.

Table 10.1: Input for Count

Keyword	Example of value	description	default
SYMMETRY	P3221	# space group name	none
CELL	57.2 33.9 68.7 90 90 120	# unit cell dims in Ångstrom	none
		# and angles in degrees	none
RESOLUTION	2.0	# resolution in Ångstrom	none
RECORD	myrecord	# file name for a brief report	history
PDB_FILENAME	abc.pdb	# file of atoms to be counted	none
BCORR	0	B-value correction	none
	commonly-used optional input for Count		
LEVELS	1.5 2. 2.5	3 levels at which to count	1. 1.5 2.

10.3 Shapes

Shapes determines the local topology at each point in a (regridded) data set. It uses the same input as Regrid (see Table 9.1). It establishes the topography in terms of one of 10 possible shapes, according to the values of 1st and 2nd derivatives of the density at a point. See Table 10.2. The indices (0 – 5) may be considered "normal"; other values are probably unphysical and indicate some problem in shape determination.

Table 10.2: Local Shapes

shape descriptor	index
uniform	0
blob	1
snake	2
saddle	3
plate	4
constriction	5
negative saddle	6
negative plate	7
negative snake (tunnel)	8
negative blob (hole)	9
"none of the above"	-1

10.4 Dphase

Dphase calculates the phase differences and the cosines of the phase differences, both weighted by amplitudes, between comparable $\overline{h} = (hkl)$ structure factors in two fcalc files. It also calculates R factors, in order to estimate amplitude differences in the two files.

For clarity, we use h in place of $\overline{h}$ and N_h for the total number of structure factors. Denoting by ϕ_h and ψ_h the phases for comparable (h, k, l) structure factors in the two files and by F_h the amplitude of (either) one of them, Dphase reports the average phase difference:

$$\sum_{h=1}^{N_h} F_h * |\phi_h - \psi_h| / \sum_{h=1}^{N_h} F_h$$

and the average cosine of the phase difference:

$$\sum_{h=1}^{N_h} F_h * cos(\phi_h - \psi_h) / \sum_{h=1}^{N_h} F_h$$

together with the number of addends in the summation. The information is reported first for all phases, then for restricted (centric) phases only. The report is prepared twice — once weighted by the amplitudes of the first fcalc file and then weighted by the amplitudes of the second. In each case, data are averaged and reported over shells of equal $1/d^2$ in (hkl) space. Dphase excludes terms for which the amplitude in either file is 0 and it excludes the (000) term. The R factors are calculated as in Solve, except that first one and then the other fcalc file serves as the "data".

Run Dphase by typing:

`eden dphase` *name sfname1 sfname2*

where *name* stands for an input parameter file name without extension `.inp`. Input will be taken from two fcalc files of structure factors, *sfname1* and *sfname2*, with extensions written out in full. Dphase expects the input parameter file, *name*`.inp`, to contain basic parameters as described in Table 3.1. Note that the two fcalc files should be similarly apodized.

It is our experience that two structure factor files whose overall phase difference is less than 20° will have physical-space counterparts that are indistinguishable when viewed with a crystallographic display program. A phase difference of 30° is just noticeable.

10.5 Distance

Distance compares real-space *.bin* files, using several measures, as described below. It may be used to compare up to 8 files at a time, reporting the distances among them in the form of a matrix.

Distance reports the rms fractional distance between pairs of input files:

$$\sqrt{\sum_{p=1}^{P}(n_p - n'_p)^2 \,/\, (\sum_{p=1}^{P}(n_p)^2 + \sum_{p=1}^{P}(n'_p)^2)}$$

.

and the absolute linear fractional distance between pairs of input files:

$$\sum_{p=1}^{P}|n_p - n'_p| \,/\, \sum_{p=1}^{P}(n_p + n'_p)/2$$

.

Distance also reports the correlation coefficient for the data in pairs of files:

$$r = \frac{\sum_{p=1}^{P}(n_p - \overline{n})(n'_p - \overline{n'})}{\sqrt{\sum_{p=1}^{P}(n_p - \overline{n})^2}\sqrt{\sum_{p=1}^{P}(n'_p - \overline{n'})^2}}$$

10.6 Variance

Variance compares a number, $M(\leq 50)$ of input binary files; it writes three output binary files: average.bin, containing the average of the M inputs at each voxel:

$$< n_p >= \sum_{m=1}^{M} n_p/M$$

sterror.bin, containing the standard error (square root of the variance) at each voxel:

$$ste(n_p) = \sqrt{\sum_{m=1}^{M}(n_p - < n_p >)^2/(M-1)}$$

and erwm.bin, the error-weighted average at each voxel:

$$erw(n_p) =< n_p >^2 /(< n_p > +ste(n_p))$$

This is a useful utility to run in conjunction with Perturbhkl: if the solution of a high-resolution fcalc file is repeatedly perturbed and solved, then finally averaged using Variance, an even better map should result. See also `stab_script` and `doit` in Appendix B

Run Variance by typing:

`eden variance` *pname b1 b2 b3 b4 b5 b6 b7 b8*

where *pname* stands for the input parameter file and *b1 b2* ... stand for the names of the binary Solve output files to be compared.

Chapter 11

Advanced Topics

11.1 Stopping Criteria for Solve Runs

There are a number of reasons why the conjugate gradient solver in Solve will quit. Some of the commoner ones will now be described. A succeesful Solve run generally ends on one of the following three conditions (where the numbers are examples only):

• "discrepancy principle satisfied"
• "Stopping - Rfac is less than 0.02"
• "getsol worked, 325 funct calls"

The discrepancy principle is a measure of the inherent accuracy of the fobs measurements (based on the σ values). Using it helps to prevent the program from overfitting the diffraction data. The Rfac stopping criterion will be effective only if you have set R_STOP, since the default is 0. Setting r_stop to a larger value can also prevent Solve from "churning"; however, if your fobs file has σ values, the discrepancy criterion generally achieves the same goal in a less arbitrary fashion. The third condition does not always signal genuine "success"; it is based on the inner workings of the complex conjugate gradient solver [5].

The most commonly encountered reason for the inner iteration loop in Solve to stop (and a new outer iteration to begin) is:

• "df/dx went down enough"

The value of $dfdx\_crit$ that triggers this message is governed by keyword DFDX_CRIT whose default is $3 * 10^{-2}$, It determines the extent to which the conjugate gradient optimizer will persist in the face of a decreasing gradient of the function being optimized. It is sometimes useful to play with its value in range 10^{-4} to 10^{-2}. Use it in conjunction with observations of the cost file, which is written when Solve is run in verbose mode.

The most common overall condition for stopping Solve is:

• "Stopping - standard deviation is not decreasing . . . "

Unsuccessful Solve runs will typically have one of the following self-explanatory reasons for ending:

• "Exceeded maximum # of iterations in getsol"

• "Dead in the water - making no progress"

(The maximum number of iterations, MAXIT, is currently 600.)

11.2 Debug Aids

There are several ways in which you can get "inside information" on the way that Solve (or another program) is working; the main method is to run Solve or Back with the verbose switch:

```
        eden -v solve run22
```

for example. In this case, Solve will produce a file, `run22.cost` that lists the cost function each time it is calculated. If the cost function has components — e.g., in an MIR run, or with physical-space constraints — each of those components is also listed in that file. There is additional output in the form of outlier reports, described in Section 4.4, if the verbose switch is set.

If you are concerned about the various (hkl) procedures that determine forbidden reflections and unique reflections, or if you just want to explore the symmetry operations that are being applied for your space group, you may turn on a (hidden) "very_verbose" switch (with upper-case `V`):

```
        eden -V solve run22
```

This will list a lot of details in your log files, including $h - k$ maps by l-slice of various masks. It will also inform you about memory allocations and deallocations during the running of the program. if you encounter problems with memory allocation and you are using double precision for all Eden's activities, you should consider switching to single precision, using the DOUBLESWITCH in the Makefile.

11.3 Other Utilities

The first thing you may notice if you type `eden` without arguments is that the general help message lists a number of programs that have been mentioned only briefly or not at all up to this point. They are mainly of interest to code developers. Nevertheless, for the record, a brief summary of each of them follows. For further information, use the help flag:

        `eden -h` *program*.

• Addmaps adds or subtracts comparable entries in two sets of real-space binary (electron/voxel) files. Note: files with the `.bin` extension, *NOT* with the `.map` extension! The input files must be compatible (of the same dimensions). As usual, you need not worry about body-centered cubic file sets as against simple cubic file sets: the program handles this distinction automatically. Addmaps adds its input maps using coefficients keyords C1 and C2 that default to 1. You may set keyword C2 to -1 in the input file, in order to subtract file2 from file1, or you may request any other linear combination of the files.

• Cadhkl (Combines Assorted Data) [1] adds, merges or eliminates comparable entries in two structure factor

[1] Thanks to CCP4 [2].

files. Entries are added by default, but only if both files contain them. As in the case of Addmaps (see above), you may enter coefficients, `C1` and `C2` for the files. You may set keyword C2 to -1 in the input file, in order to subtract file2 from file1, or you may request any other linear combination of the files. You may also merge two structure factor files if there is a keyword MODE whose value is `merge`. Cadhkl will take phases from the first file and amplitudes from the second; in this case, the second named file may be either an fcalc or an fobs file. If the value of MODE is `eliminate`, Cadhkl writes into the output file amplitudes and sigmas from the first named file if and only if the (hkl) entry is *not* in the second named file. Both input files are expected to be fobs files.

• Forth applies a Fast Fourier Transform to electron/voxel information, converting it to structure factors. The output of Forth is a file named *sname*`_forth.hkl` where *sname* stands for the binary file base name.

• Multmaps multiplies comparable entries in two sets of real-space binary (electron/voxel) files. Note: files with the `.bin` extension, *NOT* with the `.map` extension! The input files must be compatible (of the same dimensions). As usual, you need not worry about body-centered cubic file sets as against simple cubic file sets: the program handles this distinction automatically. This is a convenient way to apply a mask to another map file.

• Perturbhkl applies a perturbation to both real and imaginary parts of the structure factors of an input fcalc file. The applied perturbation is identified on the execute line in terms of a fraction (e.g. "0.2" for a 20% perturbation) and a starting seed for the random number generator. Perturbhkl is a useful tool to use in conjunction with the Variance utility, to evaluate with high precision the stability of a high-resolution Solve result. See also `stab_script` in Appendix B.

• Refine enables you to do a "re-refinement" of your Eden solution. If you feel that your final results contain really reliable phases, you can use the .map file as a basis for fine-tuning your .pdb file. This new module will optimize positions (x, y, z), B factors and occupancies of all the pdb entries to give an optimal fit to the electron densities that Eden produced in your map. Our experience with this module is very limited and we will appreciate any data that you, the users of Eden, can provide regarding its usefulness.

• Tohu reads a pdb file and transforms its data into a structure factor (fcalc) file. It regards atoms as points (i.e., it does not use atomic structure factors from the literature) but it accepts a B value for each atom. It makes appropriate use of occupancies and produces structure factors that are on an absolute scale. Tohu may be regarded as a simple-minded alternative to standard crystallographic programs with the same general purpose, such as SFALL in CCP4. It is possible to process anomalous data in Tohu if you set keyword `ANOM` to `TRUE`. In that case, Tohu will write out a file of "hydrogen" atoms at the specified positions; further processing (using Z, f', and f'') is relegated to Solve.

Appendix A

General Installation

You should have a directory, `EDEN/` with the following subdirectories: `source/` containing files with extensions `c`, `f`, `h` and `lib` plus a Makefile; `help/` containing the files invoked when Eden encounters errors or requests for help; `example1/` containing input for a trivial test problem; and `manual/` containing the PDF version of this manual.

There are several adjustments that you need to make before you can compile and load Eden. They relate both to the contents of the Makefile and to the FFTW library. First, you may need to re-install FFTW and change the location of the FFTW include and libraries. Here is the procedure that I have found to be satisfactory: when you have located and downloaded fftw-2.1.5.tar.gz, unzip it and run

```
tar -xf fftw-2.1.5.tar
```

to extract the contents. Prepare an address for your new information (e.g. `$HOME/myfft`) and then type

```
make clean
./configure --enable-type-prefix --prefix=''$HOME/myfft'' --enable-float
make
```

This will create the single-precision version of fftw, placing libraries in myfft/lib and include files in myfft/include. Then do the same without –enable-float for the double-precision library:

```
make clean
./configure --enable-type-prefix --prefix=''$HOME/myfft''
make
```

Second, you must decide whether to compile and use Eden in single- precision or double-precision or both. (Of course, if you do both, you must ”make clean” between compiling in one precision and the other.)

Third, you may wish to change the optimization level in the Makefile. And finally, you may need to change the location of the executables. There are comments in the Makefile to guide you in all these. Having made the adjustments, you should be able to compile and load Eden by issuing the commands

```
cd $EDENHOME/source
make install
```

Note that object code is first written into the `source/` directory; the word "install" transfers it to the bin/ directory. It is named "deden" or "seden" (for double- and single- versions, respectively) with "eden" being a simple script that directs you to the most recently installed version. We have encountered the following apparently harmless warning message on SGI machines while linking the object code:

```
ld:  WARNING 85:  definition of main in eden.o
preempts that definition in /usr/lib/libftn.so.
```

Recent code versions with the Refine package make use of the GSL library; we have encountered the following message – again, apparently harmless, if obscure – when linking this library on our MacOSX:

```
ld:  warning dynamic shared library:  /sw/lib/libgslcblas.dylib not
made a weak library in output with MACOSX_DEPLOYMENT_TARGET
environment variable set to:  10.1
```

If there are other (real) problems, please contact Hanna Szőke, (phone: 925-422-9248, e-mail: `szoke2@llnl.gov`).

This manual describes Eden Version 5.2; when you type `eden`, that version number should appear on your terminal. If a different version of Eden is reported, you have a mismatch between source code and manual, and the executable will *not* always behave as described here.

Appendix B

Tools

If your computer is not IEEE but has little-endian addressing and you exchange `.bin` files with users who work at big-endian computers (or vice-versa), those files must have their floating-point entries byte-swapped before they may be used as expected. To do this, look for an executable in your bin/ directory `fbyteswap` and run it with two arguments — the input file and an output file which should then replace the original. Once you have done this, you should have no further need for byte swapping.

There are other tools that may make your life easier; here is an example.

awk_hkl_to_xplor:

```
#
# AWK script to convert .hkl file in fixed (3I4,2F8.2) format to
# an X-PLOR format that EDEN can read. The script will work for
# input files in which fields lack white space separators.
#
# Run it by typing:
# awk -f awk_hkl_to_xplor ¡ [infile] ¿ [outfile]
#
# where [infile] and [outfile] stand for input & output file names.
#
{line=$0;
a=substr(line,1,4);
b=substr(line,5,4);
c=substr(line,9,4);
d=substr(line,13,8);
e=substr(line,21,8);
print “IND ”, a, b, c, “ FOBS ”, d, “ SIG ”, e }
```

Bibliography

[1] Brünger, A.T. (1992) *X-PLOR: A System for Crystallography and NMR* Version 3.1. New Haven: Yale University.

[2] *The CCP4 Suite - Overview and manual* (2004).

[3] Cowtan, K. and Main, P., *Miscellaneous Algorithms for Density Modification*. Acta Cryst. D54, 487 - 493.

[4] Creighton, T. E. (1993), *Proteins: Structure and Molecular Properties*. 2nd edition, Freeman, New York.

[5] Gill, Murray and Wright, *Practical Optimization*, pp 306-7. Academic Press, 1981.

[6] Glusker, J. P. and Trueblood, K. N., *Crystal Structure Analysis*, 1985.

[7] Goodman, D.M., Johansson, E. & Lawrence, T.W. 1993. *Multivariate Analysis: Future Directions*, edited by Rao, C.R., Ch. 11, Amsterdam: Elsevier.

[8] Hahn, Theo (ed). (1992). *International Tables for Crystallography*, 3rd edition. Vol A. Kluwer.

[9] Kleywegt, Gerard. *Uppsala Software Factory, MAPMAN Manual 1*.
See www.molsci.csiro.au/gerard/mapman_man.html.

[10] Lánczos, Cornelius, *Linear Differential Operators*, 1961, p. 132.

[11] Somoza, J.R., Szőke, H., Goodman, D.M., Béran, P., Truckses, D., Kim, S-H., & Szőke, A. (1995) *Holographic Methods in X-ray Crystallography. IV. A Fast Algorithm and its Application to Macromolecular Crystallography*. Acta Cryst. A51, 691 - 708.

[12] Szőke, A., Szőke, H. & Somoza, J.R. *Holographic Methods in X-ray Crystallography. CCP4 Daresbury Study Weekend Proceedings*
http://util.ucsf.edu/people/somoza/holography/references.

[13] Szőke, A. (1993) *Holographic Methods in X-ray Crystallography. II. Detailed Theory and Connections to Other Methods of Crystallography*. Acta Cryst. A49, 853 - 866.

[14] Szőke, H., Szőke, A., & Somoza, J.R. *Holographic Methods in X-ray Crystallography. V. Multiple Isomorphous Replacement, Multiple Anomalous Dispersion and Non-crystallographic Symmetry*. Acta Cryst. A53, 291 - 313.

[15] Szőke, A., *Use of Statistical Information in X-ray Crystallography with Application to the Holographic Method*. Acta Cryst. A54, 543 - 562.

[16] Szőke, H., Szőke, A., & Somoza, J.R. *Holographic Methods in X-ray Crystallography. VII. Spatial Target Functions* To be published.

Index