



Commission on Crystallographic Computing International Union of Crystallography

<http://www.iucr.org/iucr-top/comm/ccom/>

Newsletter No. 5, January 2005

This issue's theme:

"At Right Angles to Conventional Crystallographic reality: incommensurate structures, quasicrystals and pair distribution functions"

<http://www.iucr.org/iucr-top/comm/ccom/newsletters/>

Table of Contents

(This Issue's Editors: Simon Billinge, Gervais Chapuis, Lachlan Cranswick and Ron Lifshitz)

(Editors' warning – unless you want to kill 103 pages worth of forest – DO NOT press the “print” button. For hardcopies – you may like to only print out the articles of personal interest.)

<u>CompComm chairman's message</u> , <i>Ton Spek</i>	2	<u>Graphical and interpretation tools for difficult incommensurate and composite structures in JANA2000</u>	40
<u>Editors' message</u> , <i>Simon Billinge, Gervais Chapuis, Lachlan Cranswick and Ron Lifshitz</i>	2	<i>Václav Petříček and Michal Dušek</i>	
<u>IUCr Commission on Crystallographic Computing</u>	3	<u>Calculating the Pair Distribution Function from a Structural Model</u>	47
<u>Preliminary registration for the IUCr Computing School, Siena, Italy, August 2005</u>	4	<i>Thomas Proffen</i>	
Incommensurate structures, quasicrystals and pair distribution functions. (programming and general articles) :		Other Articles :	
<u>Procedures for the refinement of incommensurate structures using XND. Coding issues for the refinement of incommensurate structures.</u>	5	<u>Refinement in Crystals</u>	51
<i>Jean-François Béar and Gianguido Baldinozzi</i>		<i>Richard Cooper and David Watkin</i>	
<u>A Program Package for Aperiodic Tilings</u>	10	<u>cctbx news: Phil and friends</u>	69
<i>Uwe Grimm</i>		<i>Ralf W. Grosse-Kunstleve, Pavel V. Afonine, Nicholas K. Sauter and Paul D. Adams</i>	
<u>DIMS (Direct-methods program for solving Incommensurate Modulated Structures) on the VEC platform</u>	16	<u>Computing the Z-matrix for global optimisation</u>	92
<i>Hai-fu Fan</i>		<i>Kenneth Shankland</i>	
<u>DIMS (Direct-methods program for solving Incommensurate Modulated Structures) /VEC applications</u>	24	<u>Calls for contributions to Newsletter No. 6</u>	103
<i>Hai-fu Fan</i>			
<u>Collection and visualization of single crystal data of incommensurate crystals</u>	28		
<i>Rob Hooft</i>			
<u>Visualization and Analysis of Single Crystal Time-of-Flight Neutron Scattering Data using ISAW</u>	32		
<i>Dennis Mikkelsen, Arthur J. Schultz, Ruth Mikkelsen and Thomas Worlton</i>			

CompComm Chairman's Message

This newsletter has again managed to bring together a large number of very relevant and interesting computing related articles. The focus this time is on the computational aspects of incommensurate and related non-standard structures.

David Watkin's article teaches us how to translate 'Cambridge speak' (SHELXL) into 'Oxford speak' (Crystals). Ralf Grosse-Kunstleve informs us about the latest extensions to the Computational Crystallography Toolbox.

We expect to close the term of the current IUCr Computing Commission with a very interesting 'classical' computing school addressing state-of-the-art (hands-on) software development and of particular relevance for young scientists interested in crystallographic computing. Details can be found below.

Ton Spek, Chairman of the IUCr Computing Commission, (a.l.spek@chem.uu.nl)

From the Editors of Newsletter No. 5

Since more than three decades, crystallographers have been faced with new challenging crystalline material with structures incompatible with the classical view of crystals with three dimensional periodicity. These new materials includes incommensurately modulated and composite structures, quasicrystals with icosahedral or dodecagonal symmetry to cite only the most representative examples of aperiodic structures as they are presently called. In most cases, these new structures are best described by embedding them in space of up to six dimensions. This approach is justified by the fact that periodicity can be recovered although in higher dimension.

The rapid evolution of this field is not only due to the innovative theoretical approach of the so called superspace symmetry but also to the enormous worldwide efforts on software development. This issue includes a range of articles on techniques that can explain diffraction data where the most appropriate model may not fit into an ordered convenient commensurate cell: Incommensurate Structures, Quasicrystals and Pair Distribution Functions. Besides encouraging exchange of ideas within different communities, we hope it might encourage crystallographers, who may prefer to deal only with ordered commensurate cells, to take these style of problems out of the "too unusual draw" and onto their diffractometers and transmission electron microscopes.

Simon Billinge, Gervais Chapuis, Lachlan Cranswick and Ron Lifshitz

(billinge@pa.msu.edu ; gervais.chapuis@epfl.ch ; lachlan.cranswick@nrc.gc.ca ; ronlif@tau.ac.il)

THE IUCR COMMISSION ON CRYSTALLOGRAPHIC COMPUTING - TRIENNium 2003-2005

Chairman: Prof. Dr. Anthony L. Spek

Director of National Single Crystal Service Facility,
Utrecht University,
H.R. Kruytgebouw, N-801,
Padualaan 8, 3584 CH Utrecht,
the Netherlands.
Tel: +31-30-2532538
Fax: +31-30-2533940
E-mail: a.l.spek@chem.uu.nl
WWW: <http://www.cryst.chem.uu.nl/spea.html>

Professor I. David Brown

Brockhouse Institute for Materials Research,
McMaster University,
Hamilton, Ontario, Canada
Tel: 1-(905)-525-9140 ext 24710
Fax: 1-(905)-521-2773
E-mail: ldbrown@mcmaster.ca
WWW: http://www.physics.mcmaster.ca/people/faculty/Brown_ID.html

Lachlan M. D. Cranswick

Neutron Program for Materials Research (NPMR),
National Research Council (NRC),
Building 459, Station 18, Chalk River Laboratories,
Chalk River, Ontario, Canada, K0J 1J0
Tel: (613) 584-8811 ext: 3719
Fax: (613) 584-4040
E-mail: lachlan.cranswick@nrc.gc.ca
WWW: <http://neutron.nrc.gc.ca/peep.html#cranswick>

Dr Vincent Favre-Nicolin

CEA Grenoble
DRFMC/SP2M/Nano-structures et Rayonnement Synchrotron
17, rue des Martyrs 38054 Grenoble Cedex 9
38054 Grenoble Cedex 9 – France
Tel: (+33) 4 38 78 95 40
Fax: (+33) 4 38 78 51 97
E-mail: vincefn@users.sourceforge.net
WWW: <http://objcryst.sourceforge.net/>

Dr Ralf Grosse-Kunstleve

Lawrence Berkeley Lab
1 Cyclotron Road,
BLDG 64R0121,
Berkeley, California, 94720-8118, USA.
Tel: 510-486-5909
Fax: 510-486-5909
E-mail: rwgk@yahoo.com
WWW: <http://cci.lbl.gov/>

Prof Alessandro Gualtieri

Università di Modena e Reggio Emilia,
Dipartimento di Scienze della Terra,
Via S.Eufemia, 19,
41100 Modena, Italy
Tel: +39-059-2055810
Fax: +39-059-2055887
E-mail: alex@unimore.it
WWW: <http://www.terra.unimo.it/mineralogia/gualtieri.html>

Prof Ethan A Merritt

Department of Biological Structure
University of Washington
Box 357420, HSB G-514
Seattle, Washington, USA
Tel: 206 543 1861
Fax: 206 543 1524
E-mail: merritt@u.washington.edu
WWW: <http://www.bmsc.washington.edu/people/merritt/>

Dr. Simon Parsons

School of Chemistry
Joseph Black Building,
West Mains Road,
Edinburgh, Scotland EH9 3JJ, UK
Tel: +44 131 650 5804
Fax: +44 131 650 4743
E-mail: s.parsons@ed.ac.uk
WWW: <http://www.chem.ed.ac.uk/staff/parsons.html>

Dr. Bev Vincent

RigakuMSC
9009 New Trails Dr,
The Woodlands, Texas 77381-5209, USA
Tel: 281-363-1033
Fax: 281-364-3628
E-mail: brv@RigakuMSC.com
WWW: <http://www.rigakumsc.com/>

Consultants

Dr David Watkin

Chemical Crystallography,
Oxford University,
9 Parks Road,
Oxford, OX1 3PD, UK.
Tel: +44 (0) 1865 272600
Fax: +44 (0) 1865 272699
E-mail: david.watkin@chemistry.oxford.ac.uk
WWW: <http://www.chem.ox.ac.uk/researchguide/djwatkin.html>

Dr Harry Powell

MRC Laboratory of Molecular Biology,
Hills Road, Cambridge, CB2 2QH, UK.
Tel: +44 (0) 1223 248011
Fax: +44 (0) 1223 213556
E-mail: harry@mrc-lmb.cam.ac.uk
WWW: <http://www.mrc-lmb.cam.ac.uk/harry/>

Now Accepting Preliminary Registrations via the school website



IUCr Commission on Crystallographic Computing Siena 2005: Crystallographic Computing School



**Certosa di Pontignano,
University of Siena, Italy
18th - 23rd August 2005**

(just prior to the [Florence IUCr 2005 congress](#))

<http://www.iucr.org/iucr-top/comm/ccom/siena2005/>

School Organisers: Prof Anthony Spek (Utrecht), Prof. Marcello Mellini (Siena), Prof. Alessandro Gualtieri (Modena), Dr Harry Powell (Cambridge), Lachlan Cranswick (NRC Chalk River)

Consultants: Dr David Watkin (Oxford), Dr Simon Parsons (Edinburgh)



The City

Siena is described as one of the finest examples of a Medieval city. It is in the Italian province of Tuscany and has direct bus connection to Florence (1 hour) and Rome (3 hours).



The Venue

The Certosa di Pontignano has its origins as a medieval 14th century monastery. It is now run by the University of Siena. Attractively placed on the top of a hill, it is surrounded by vineyards; with a direct view to the town of Siena, and a famous Chianti winery.

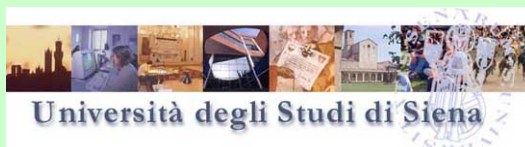


School Aims

To have the crystallographic computing experts of the present, help train and inspire a generation of experts for the future. This will be achieved by the use of an excellent (and full) [program of lectures](#), workshops and projects.

Each day of the school is focussed on a different theme:

- “principles & methods”
- “joining things together”
- “crystallographic implementations”
- “selected topics in crystallography”
- “special methods”



Procedures for the refinement of incommensurate structures using XND.

Coding issues for the refinement of incommensurate structures.

Jean-François Bérar⁽¹⁾ and Gianguido Baldinozzi⁽²⁾

1. Laboratoire de Cristallographie, CNRS, BP 166, F 38042 Grenoble Cedex, France.

berar@grenoble.cnrs.fr ; http://www-cristallo.grenoble.cnrs.fr/Le_Personnel/CVs/BERAR.html and

2. SPMS, CNRS ECP, F 92995 Chatenay-Malabry Cedex, France.

gianguido.baldinozzi@ecp.fr

Introduction

The Rietveld program *xnd* [1] was first written in the late 80's to take full profit of data collected with high resolution laboratory diffractometers. *Xnd* handles complex diffraction patterns including : data sets contain multiple wavelength contributions, multiple phases, handled defining a proper structure or as a simple set of parasitic peaks, patterns recorded with irregular angular steps, joint refinements of multiple data sets (x-ray, neutron, single crystal), constrained refinements of multiple data sets of a given sample in a changing environment (field, temperature, pressure, ...) ; the program also handles a variety of experimental setups and their proper lineshapes and absorption corrections.

The proper use of *xnd* requires a good knowledge of diffraction experiments and the program is not designed for a black box usage. For instance, the way the profile lineshape is defined finds its roots in this more complex approach to diffraction: in fact, each wavelength and each phase in a high resolution diffraction pattern have *a priori* different profile functions and a specific behaviour. The overall profile functions are then described by the convolution of these individual contributions coming from the experimental geometry and from the intrinsic sample broadening. They are efficiently expressed as Voigt functions, allowing an accurate lineshape modelling during a reasonable calculation time [2, 3]. Within this approximation, the lineshape contributions can be described by Lorentzian and Gaussian function widths with a proper angular dependence that can be straightforwardly related to a particular physical or instrumental origin. Following the same *leit motiv*, preferred orientation effects were also taken into account using a polynomial expansion on the spherical harmonic basis. These functions were also used to model the sample anisotropic broadening due to finite crystallite size or microstrain effects.

The proper definition of the lineshape is the necessary base for handling the efficient refinement of the diffraction pattern of a modulated phase and, in the following discussion, we would like to stress the main features of such a refinement with *xnd*. In the case of composite structures the key points developed in the discussion are similar but several input parameters have to be modified.

Incommensurately modulated phases.

Most of the structural studies of incommensurately modulated phases are developed using single crystal diffraction data. Nevertheless, many compounds are not easily synthesized as single crystals or the interesting phases present complex polydomain structures. Therefore, the study of the structure by single crystal techniques becomes very complex, or even not possible. The 4D formalism for mono-incommensurately modulated phases (superspace group symmetry, intensity and positioning of satellite reflections) was implemented in *xnd* [4]. In *xnd* the position u_i^μ for the atom μ in the modulated structure along the coordinate i is given by :

$$u_i^\mu(x_4^\mu) = \sum [S_{n,i}^\mu \sin(2\pi n x_4^\mu) + C_{n,i}^\mu \cos(2\pi n x_4^\mu)]$$

where x_4^μ is the variable describing the average position of the atom μ in the internal subspace defined by the modulation vector \mathbf{q} and orthogonal to the euclidean space ($x_4^\mu = \mathbf{q} \cdot \mathbf{r}^\mu$, \mathbf{r}^μ being the average position of this atom).

Another advantage is represented by the possibility to refine simultaneously x-ray and neutron diffraction data, taking advantage of the different atomic contrast and resolution available from these probes. Even in the early stages of the refinement, it is generally easier to decode the four dimensional Fourier maps obtained from x-ray diffraction as they are dominated by the heavier scatterers and there are less spurious maxima because of the atomic form factor shape. On the other hand, neutron Fourier maps give more details on the atomic positions and are very useful in the later stages of the refinements.

This complementary use of the different probes is very powerful for the analysis of the structures of oxide compounds. In particular, the refinement of the incommensurately modulated perovskites-type structures is a challenging problem as (often) the onset of the modulated phase takes place at a ferroelastic phase transition. Moreover, light and heavy scatterers are generally present in these structures (ferroelectrics, superconductors ...).

The refinement of incommensurate phases can be considered a powerful tool for localizing the structural disorder affecting a compound [5, 6]. Therefore, it is possible to get a better insight of complex and defective phases. This is particularly important since the small differences between the ideal and *real* structure of crystals are often responsible for the onset of interesting physical properties. This is for instance the case of oxygen stoichiometry in superconductors, of correlations between cation displacements in ferroelectrics, ...

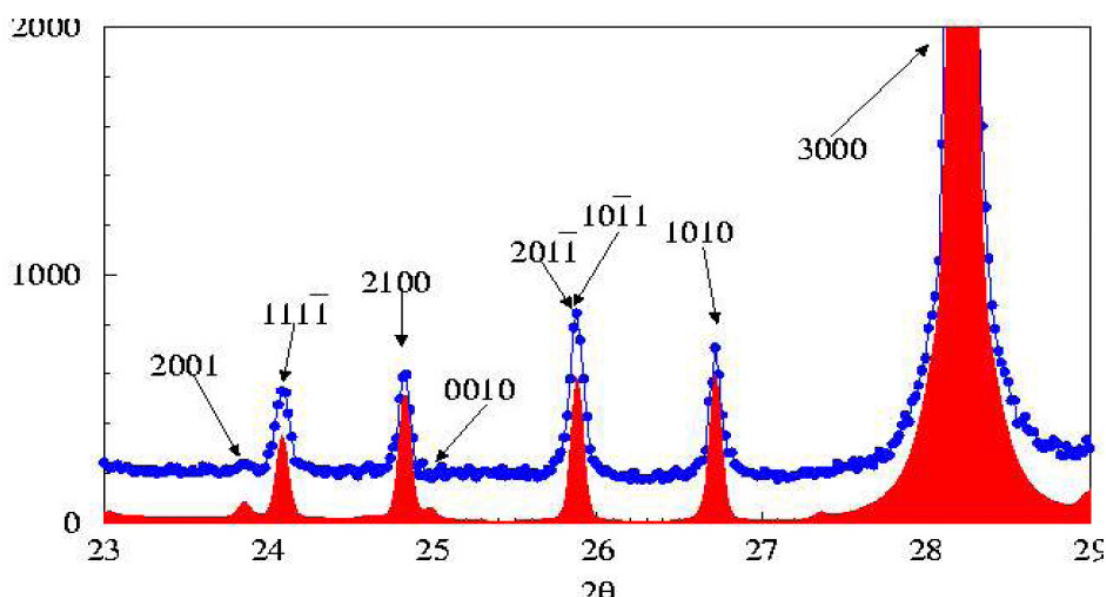


Figure 1: Example of refinement: x-ray diffraction pattern of the modulated $Ba_{0.85}Ca_{2.15}In_6O_{12}$ phase [5].

In incommensurate structure refinements it is necessary to distinguish the different satellite orders so it is useful to define different R_I (R_F) agreement factors for each set of reflections (Fig. 1). In general, satellite peaks have low peak to background ratios; therefore, the effect of the background noise on the estimated integrated intensity of the satellites is increased and the R_I factor will be generally larger for these sets of reflections, even if the structural model is good. A second problem consists in the frequent overlapping of the satellite peaks with intense average structure main peaks; in this case, a small error on the main peaks and in the description of their profile will strongly affect the estimation of the experimental intensities of the satellites. The importance of a very good adequation between experimental and calculated lineshapes must be emphasized and, in this domain, *xnd* offers a large choice of functions and combinations of functions, with angle dependent parameters to optimize the refinement.

Recommended procedure.

The refinement of a modulated phase is generally a complex task because it generally involves the problem of a least-square matrix where the larger number of structural parameters of the refinement have an uneven weight. A robust approach is therefore recommended. During the progression of the refinement, the actual importance of the different parameters must be examined and it is important to study their eventual correlations. Once the refinement is stable, the complexity of the structural model can be gradually increased, reducing the risks of large instabilities in the refinement. The main drawback of this approach is that it may drive the refinement towards a local minimum but, often, the modulation waves of lowest order are the leading terms to explain the intensities of the satellite reflections. Nevertheless, it is a good policy to check for the existence of local minima and to assess the likelihood of the model, like in any complex refinement. In the following, a simple example of modulated structure is analysed. It concerns the refinement of the neutron diffraction pattern of the incommensurate phase of the ordered perovskite Pb_2CoWO_6 [7]; the code shown below correspond to *xnd* release 1.27.

- **The first step** consists in refining all what can be attempted without going into the incommensurate phase : background, parasitic phases, main structure and line profile. At this step the input file does not differ from a standard one, we now have to describe the incommensurate phase. The analysis of the anisotropic thermal displacement parameters can provide a useful hint for the initial amplitudes of the modulation waves.
 - **Incommensurate vector and symmetry** : this essential information is often obtained by transmission electronic microscopy, from the analysis of the microdiffraction pattern of a single domain region of the sample. In this phase of Pb_2CoWO_6 the satellite spots are observed near the main spots forbidden by the I centring and they can be indexed by the modulation vector $\mathbf{q} = \alpha \mathbf{a}^* + \gamma \mathbf{c}^*$. The analysis of the systematic extinctions leads to the planar monoclinic superspace group $I2/m(\alpha 0 \gamma)0s$. It is generally possible to refine precisely the components of the modulation vector. Therefore, a rough estimate of these components by electron diffraction is generally a very good starting point.
1. **Symmetry data** : in *xnd* the 3+1D space groups are introduced using the basic 3D space group and the effect of the symmetry operation on the modulation vector as another space group. Then we find in the header of the structure file two space group blocks, one for the 3D part (the usual space group) and another one for the complementary part being identified by “*”.

```
                                I2/M # standard group
M 1 1 2
X, Y, Z;  X, -Y, Z;

*_I2/M # complementary part on the 4th coordinate
* 1 0 2
X;  X+1/2;
```

- **Declaration of the incommensurate phase :** this is done by specifying the new nature code which requires “MODUL” and “VECTOR” keys; it is also necessary to specify the “SETS” of satellites that are needed for accounting of the observed satellites in the “CRYSTAL” block. The additions to the average structure block are highlighted in the following :

```
@PHASE  TITLE = 'phase II  modulated (incom)' ; NATURE = -5 ;

@SCALE
      0.0027132651      1                      0      0

@CRYSTAL  SYMGRP = I2/M ; ATOM = 5 ; SETS = 3

@CELL
      0      0                      7.9601831      1
      5.6777182      1                      5.6963921      1
      90      0                      90.049103      1
      90      0

@MODUL  ORDER = -50 ; SYMGRP = *_I2/M
```

- **Introduction of the satellite reflections sets :** they are used to generate only the necessary satellite orders. This description in separate sets allows, when necessary, to define different line profile functions for each of them. They are followed by the vector definition which can be fitted.

```
@SET_0  LOOP = 'HKL,M=0'

@SET_1  LOOP = 'HKL,M=1,-1'
# @SET_2  LOOP = 'HKL,M=2,-2'    ## skipped comment

@VECTOR
      0.89247602      1                      0      0
      0.17110361      1
```

- **Atoms and Fourier terms :** the modulated atoms are identified by “CASE”. The following input generate the output reproduced after in which “0_S” means a *sin* component associated with 1st order, “1_C” being a *cos* component of 2nd order.

```
@ATOM  NAME = PB ; CHEM = LEAD ; CASE = 8
@COORD
      0.25192171      1                      0      0
      0.4985429      1                      0.5      0
      1.9922185      1

@MODUL
      0      0                      0      0
      0      -1                      -0.036814742      1
      0      0                      0      0
      0      0                      0      0
      0.0014000313      1                      -0.0010034939      1
      0      0                      0      0
      0      -1                      0.0017912069      1
      0      0                      0      0
```



```

PB LEAD 8: ( B ISO MODULATED=2 ) COORD : 5 parameters : expansion order 1 (size 1)
      X      0.252946      1 ( 40 1.000000)      Y      0      0 (      )
      Z      0.498337      1 ( 41 1.000000)      Oc      0.5      0 (      )
      Biso      1.86584      1 ( 42 1.000000)

PB generates 4(8/2) sites corresponding to 4.00(0.50*8) atoms in the cell
The chemical occupancy is then 1.000
PB : 16 parameters : expansion order 1 (size 1)
0 S x      0      0 (      ) 0 C x      0      0 (      )
0 S y      0 -1 ( 0 1.000000) 0 C y      -0.0389692      1 ( 43 1.000000)
0 S z      0      0 (      ) 0 C z      0      0 (      )
0 S Oc      0      0 (      ) 0 C Oc      0      0 (      )
1 S x      0.00337146      1 ( 44 1.000000) 1 C x      -0.00132127      1 ( 45 1.000000)
1 S y      0      0 (      ) 1 C y      0      0 (      )
1 S z      0 -1 ( 0 1.000000) 1 C z      -0.00494137      1 ( 46 1.000000)
1 S_Oc      0      0 (      ) 1 C_Oc      0      0 (      )

```

In a first step, to speed up the calculation, it may be useful to introduce only the first order satellites; often, the $n+1$ order wave still gives a valuable contribution to the intensities of n -order satellites. Therefore, it is sometimes necessary to introduce second order atomic displacements even if the second order satellite reflections are very weak because the refinement of the intensities of first order satellites are sensibly improved. Nevertheless, it is very difficult to determine the phase of the modulated displacements of a given wave order when the satellites of this same order are very weak or missing.

Conclusion.

The reliable refinement of modulated structures can not be considered a straightforward task. During the different stages of the refinement, it is generally useful to calculate four dimensional Fourier maps. They can be obtained for instance with the program JANA [8]; the integrated intensities needed as input file can be easily extracted from the (hkl) file generated by *xnd*, using a very simple *awk* script. The input in JANA is also very useful to draw the interatomic distances between atoms in the different sections of the modulated crystal.

References:

- [1] Bérar J-F and Garnier P, *Accuracy in powder diffraction, APD 2nd Conference*, **846**, Gaithersburg, USA, NIST (1992).
- [2] Baldinozzi G, Sciau Ph, Pinot M and Grebille D, *Acta Crystallogr. B* **51** (1995) 668
- [3] Seshadri R, Martin C, Maignan A, Hervieu M, Raveau B and Rao C N R, *J.Mater.Chem.* **6** (1996) 1585
- [4] Baldinozzi G., Grebille D and Bérar J-F, *Proceedings of Aperiodic 97, World Scientific*
- [5] Baldinozzi G., Goutenoire F, Hervieu M, Suard E and Grebille D, *Acta Crystallogr. B* **52** (1996) 780
- [6] Baldinozzi G., Grebille D, Sciau Ph, Kiat J-M, Moret J and Bérar J-F, *J. Phys.: Condens. Matter* **10** (1998) 6461
- [7] Baldinozzi G., Calvarin G., Sciau Ph., Grebille D. Suard E., *Acta Crystallogr. B* **56** (2000) 570
- [8] Petricek,V., Dusek,M. & Palatinus,L.(2000). *Jana2000. The crystallographic computing system*. Institute of Physics, Praha, Czech Republic

A Program Package for Aperiodic Tilings

Uwe Grimm

Applied Mathematics Department, The Open University, Walton Hall, Milton Keynes MK7 6AA, United Kingdom; E-mail: u.g.grimm@open.ac.uk ; WWW: <http://mcs.open.ac.uk/ugg2/>

We describe a package of Mathematica programs, originally devised for summer schools on aperiodic order and quasicrystals, which give an introduction to the construction of aperiodic tilings. The programs explore several approaches to generate aperiodic tilings, concentrating on one-dimensional and two-dimensional examples which can easily be visualised.

I Introduction

Quasicrystals, first discovered by Shechtman in 1982 (see Shechtman et al. 1984), are aperiodically ordered solids which typically display crystallographically forbidden symmetries; see e.g. Stadnik 1999, Suck et al. 2002, Trebin 2003 for recent collections of review articles. Their structure is usually modelled in terms of an aperiodic tiling of space, which plays the role of the lattice for a conventional crystals, compare e.g. Janot 1994, Senechal 1995. Apart from their use as toy models for quasicrystals, aperiodic tilings also are aesthetically appealing, and feature in some computer generated artworks. The paradigms of planar quasiperiodic tilings are the celebrated Penrose tiling (see Penrose 1974) and the octagonal Ammann-Beenker tiling (see Grünbaum et al. 1987, Ammann et al. 1992), which predate the experimental discovery of quasicrystals. A patch of the octagonal tiling, which consists of squares and rhombi, is shown in Figure 1. The infinite tiling is repetitive, in the sense that any patch occurs within the tiling over and over again, but non-periodic, which means that there is no translation that maps the tiling onto itself. This structure is pure point diffractive, i.e., the diffraction pattern of, say, point scatterers placed on the vertices of the tiling is pure point, and it furthermore has perfect eightfold rotational symmetry; see e.g. Baake 2002 for more details on the mathematical background.

The Ammann-Beenker tiling can be constructed from the two prototiles, the square and the rhombus, taking into account the arrow decorations of edges and corners as shown in Figure 1. Imposing the restrictions, known as *matching rules*, that markings have to agree on edges and that corner markings have to form complete arrows (or “houses”) at the vertices, enforces any infinite tiling obeying these rules to be aperiodic (sometimes the marked tiles themselves are referred to as *aperiodic* because all possible tilings of space with these tiles are necessarily non-periodic).

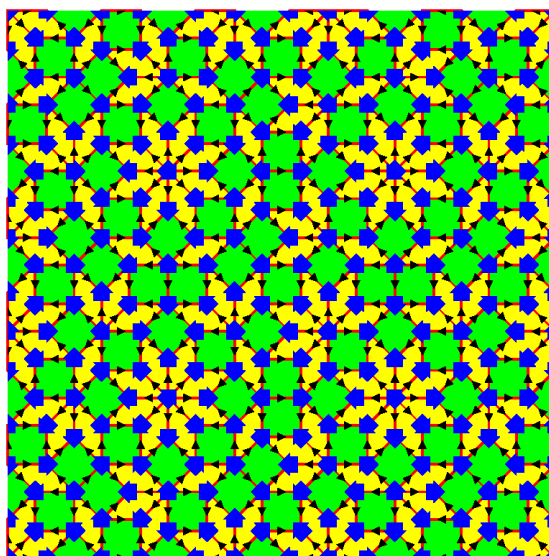


Figure 1: A patch of the octagonal tiling of squares and rhombi, with matching rules marked by the black arrows along the edges and the large blue arrows (or “houses”) at the vertices.

In this article, we describe a set of computer programs which provide an introductory account of the construction of aperiodic tilings, detailing several different approaches such as inflation, or projection from higher-dimensional periodic lattices. The programs, most of which were originally produced for a summer school on quasicrystals held in Chemnitz in 1997, have also been used at a summer school on Aperiodic Order in Edmonton in 2000, a further summer school on Computational Statistical Physics in Chemnitz in 2000, and, recently, at the Royal Society Summer Science Exhibition 2004 in London, which was mainly aimed at a general public audience.

The programs are written in the algebraic computer package *Mathematica* (a registered trademark of Wolfram Research) which provides all basic operations that are needed as well as advanced graphics tools to visualise the results. While this makes it relatively easy to play with the programs, it unfortunately also means that anyone who wants to use the package will need access to a computer that has *Mathematica* installed on it. The Aperiodic Tilings program package, which can be downloaded at <http://mcs.open.ac.uk/ugg2/AperiodicTilings/>, also contains an introduction to the use of *Mathematica*, so it is not necessary to be familiar with this algebraic computer package to explore the programs, although it will make it easier. Time permitting the author intends to add additional programs in the future.

Apart from the introductory notebook, the programs consist of two parts - the actual program code (file names with extensions "m") in form of *Mathematica* packages, compare Maeder 1990, and the interactive front-end files (file names with extensions "nb") in form of *Mathematica* notebooks. Notebook files may be slightly different depending on the actual version of *Mathematica* used; although there is usually no problem with compatibility, different versions are supplied in order to avoid the necessity of conversion. To use a program, open a notebook in *Mathematica*, and load the corresponding package file (which needs to be copied to the same location).

II Construction of Aperiodic Tilings

There are several standard approaches to construct aperiodic tilings; not all tilings will allow all of these – although the most popular tilings are those with all “magic” properties and hence can be constructed in a number of different ways.

We already mentioned the matching rule approach above. The matching rules for the Ammann-Beenker tiling are given by the arrow decorations of edges and corners, such that the decorated tiles can form only aperiodic tilings that are legitimate. One might expect that this approach holds an intuitive clue for the formation of quasicrystals in Nature, in that matching rules might be thought to mimic interactions between atoms or atomic clusters in quasicrystals; however, this is not so simple because matching rules do not constitute growth rules. The program **PenrosePuzzle** may show you why that is so. It allows you to create a part of a Penrose tiling by assembling pieces like in a jigsaw puzzle, making sure that you obey the matching rules as you go along. If you are not already too familiar with this tiling, it is likely that, sooner or later, you will run into a situation where you encounter the problem that no further tile fits at a certain position, which means that you must have made a "mistake" somewhere along the way, and the patch you constructed is not actually a part of an infinite Penrose tiling, which means you have to retreat and try again. Assembling a tiling by trial and error in this way is cumbersome and very slow. The problem is that there is no local way to tell you which tile you have to add at a given place (provided there is a choice), and since atoms or atomic clusters that assemble to form a solid do not know either, matching rules do not provide a proper explanation of quasicrystal growth.

Another method that is easier to implement is based on *inflation*. Essentially, inflation/deflation symmetry of a tiling gives you a recipe how to dissect the basic tiles into a number of smaller copies of themselves, such that repeated application of dissection with an appropriate length rescaling produces an aperiodic tiling. For instance, for the Ammann-Beenker tiling an inflation rule is known, and the program **OctagonalTiling** of the package contains an implementation of it. Other examples constructed via this

approach are, besides one-dimensional chains covered in the program `FibonacciChain`, the so-called chair and sphinx tilings, in the corresponding programs `ChairTiling` and `SphinxTiling`.

Inflation method for the pinwheel tiling

As an example, we here consider the so-called `PinwheelTiling`, see Radin 1999. The inflation rule is implemented in the program `PinwheelTiling`. As the chair and sphinx tilings, it only consists of a single tile, with side length ratios $1 : 2 : \sqrt{5}$, which however appears in infinitely many orientations. The inflation rule is shown in Figure 2; it dissects the original right-angled triangle into five congruent copies, with side lengths scaled by a factor $1/\sqrt{5}$ with respect to the original triangle.

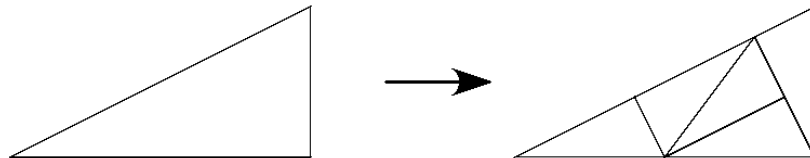


Figure 2: Inflation rule for the pinwheel tiling.

A repeated inflation of an initial patch consisting of two triangles produces the patches shown in Figure 3.

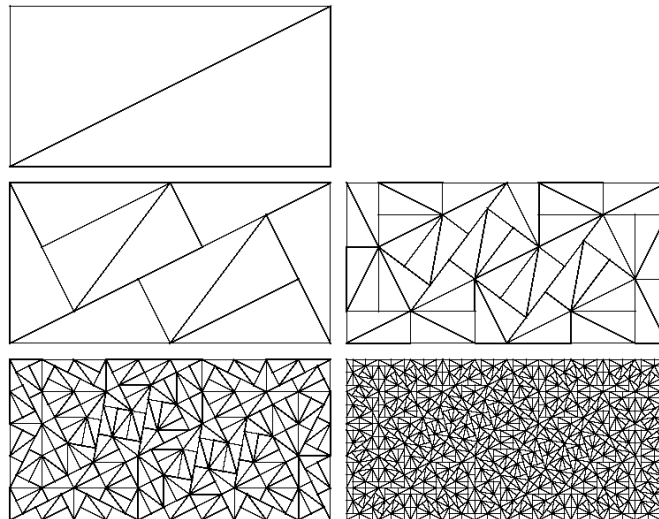


Figure 3: Patches of the pinwheel tiling obtained by repeated inflation.

The way this is implemented in *Mathematica* is as follows. The tiling consists of a list of tiles, which in turn are lists of their three vertices. The main part is the inflation of a single tile, which is done by defining a function `TileInflation` which, when acting on a list representing a single tile, produces a list containing the five dissected tiles,

```
TileInflation[{tilevertex1_,tilevertex2_,tilevertex3_}] :=
  ScaleFactor*{ {#1,#5,#4},
    {#4,#7,#2},
    {#4,#7,#6},
    {#6,#5,#4},
    {#2,#6,#3}} & [tilevertex1,
    tilevertex2,
    tilevertex3,
    tilevertex1/2+tilevertex2/2,
    3*tilevertex1/5+2*tilevertex3/5,
    tilevertex1/5+4*tilevertex3/5,
    tilevertex1/10+tilevertex2/2+2*tilevertex3/5]
```

which are then rescaled by multiplying by **Scalefactor** which is set to $\sqrt{5}$. The seven quantities in the square brackets are the seven vertices that form the five new triangles (compare Figure 2), including the three vertices of the triangle we started from; and the five lists with elements #1, #2, etc. pick out the vertices of the respective triangles. Note that in order to work correctly the vertices of the original triangle have to be specified in the correct order. The function **TileInflation** is then applied to a list of tiles by “mapping” it repeatedly over the elements of the list, which can be done as follows,

```
Inflation[tiling_List,
  num_Integer:1] /; num>=0 :=
  Nest[Flatten[Map[TileInflation,#],1]&,tiling,num]
```

Here, the second argument **num** is constrained to be an integer, and it is checked that it is non-negative, before the function **TileInflation** is mapped over the list **tiling** using the **Nest** command. The additional command **Flatten** is included to prevent the proliferation of levels of lists within lists, so the resulting tiling is again a list of its tiles.

The remaining parts of the program **PinwheelTiling** provide a function to plot the resulting tiling. This includes a function **PlotColorTiling** which colors all tiles according to their orientation, as shown in Figure 4.

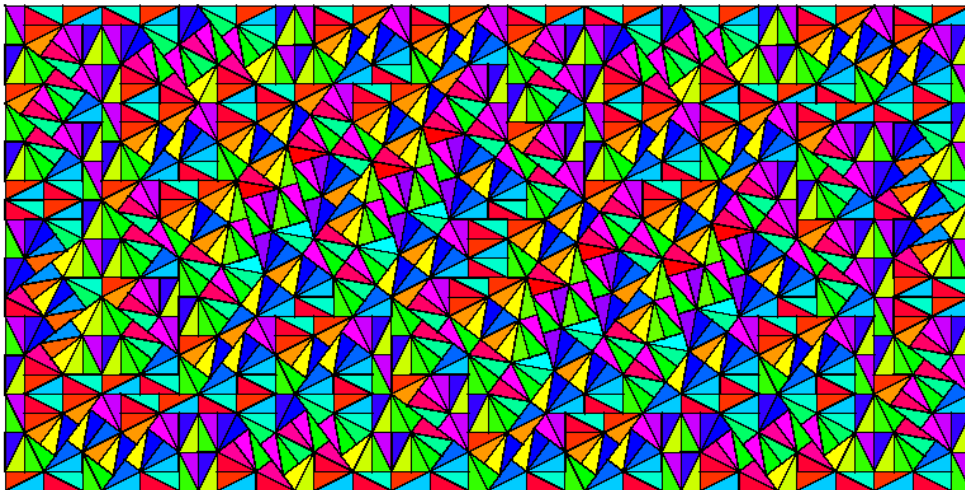


Figure 4: *Pinwheel tiling with tiles colored according to their orientations.*

Projection and grid methods

The other commonly used approach to construct aperiodic tilings is based on projection of a certain part of a higher-dimensional periodic lattice. The structures derived in this way are known as cut-and-project sets or model sets; see Baake 2002 for details. Again the program **FibonacciChain** shows how this works in the one-dimensional setting, using the ubiquitous example of the Fibonacci chain which can be obtained as a projection from a strip of the two-dimensional square lattice.

The program **OctagonalTiling** contains an implementation of the projection method for the Ammann-Beenker tiling. In this case, the periodic lattice is the integer lattice in four dimensions, so cannot be easily visualised. In this four-dimensional space, two orthogonal two-dimensional spaces are chosen, one corresponding to the “physical” (sometimes also called “parallel”) space which contains the tiling, the orthogonal complement is known as the “internal” (sometimes also called “orthogonal” or “perpendicular”) space. The internal space projection of the lattice is used to select the lattice points which are to be projected to form the tiling. For the Ammann-Beenker tiling, all lattice points in four-dimensional space whose projection on the internal space falls into a regular octagon are selected, and

their projections on the physical space form the actual tiling. By relating the aperiodic tiling to a higher-dimensional periodic lattice, the projection approach explains why these tilings have pure point diffraction patterns, although this has only recently been proven in a general setting, see Schlottmann 2000. The rotational symmetry stems from a particular choice of the physical space that retains the eightfold rotational symmetry that is present in the four-dimensional integer lattice. The projection in internal and physical space forming a patch of the Ammann-Beenker tiling is shown in Figure 5.

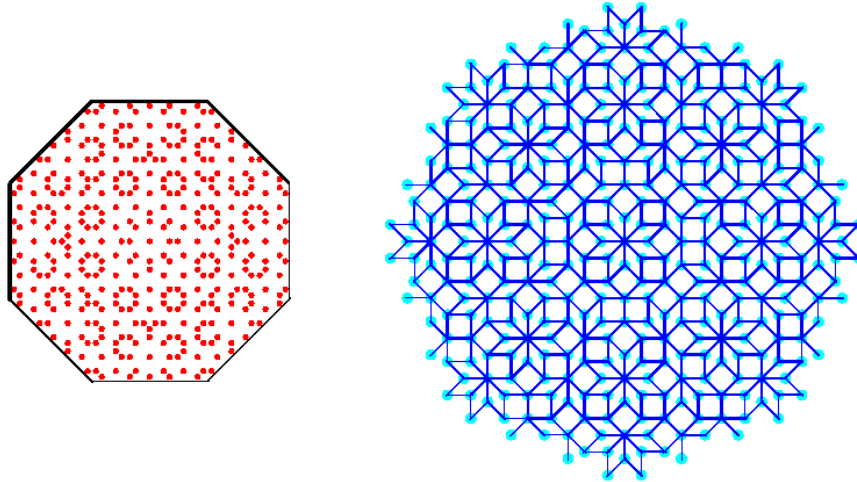


Figure 5: *Projection in internal (left) and physical (right) space. Note that the two projections are not to scale. The lattice points projected are those for which the internal projections, shown as red dots, fall inside the regular octagon. Their projections in physical space are the vertices of an Ammann-Beenker tiling.*

Closely related to this approach is the *grid method* pioneered by de Bruijn (see de Bruijn 1981), where an n -fold rotationally symmetric aperiodic tiling is constructed by dualizing a grid obtained from intersecting sets of equidistant parallel lines, rotated with respect to each other by multiples of $360^\circ/n$. The program **GridMethod** implements this for arbitrary symmetries n . Choosing $n=4$ once more yields the Ammann-Beenker tiling, as can be seen from Figure 6.

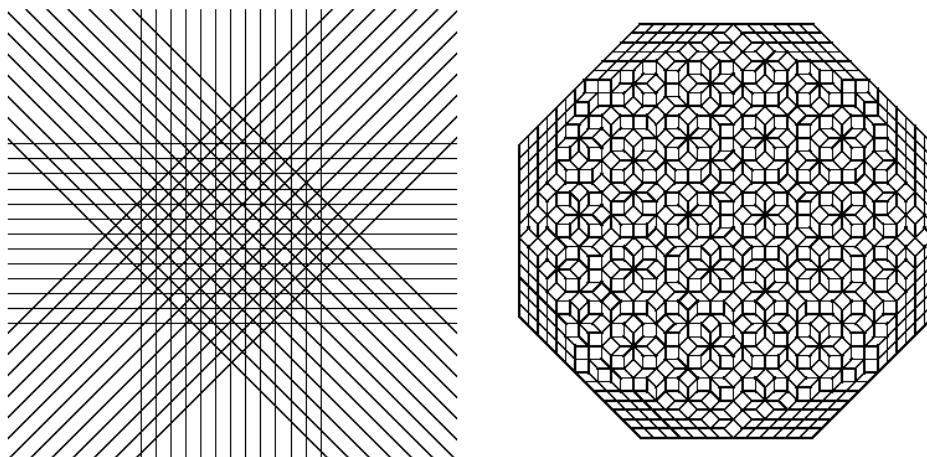


Figure 6: *De Bruijn grid and dual tiling with eightfold symmetry. The patch in the inside belongs to an Ammann-Beenker tiling, the regular parts near the boundary stem from regions where only some of the grid lines intersect.*

As mentioned, this method can be applied to other symmetries. Just for fun, an example with 13-fold symmetry is shown in Figure 7.

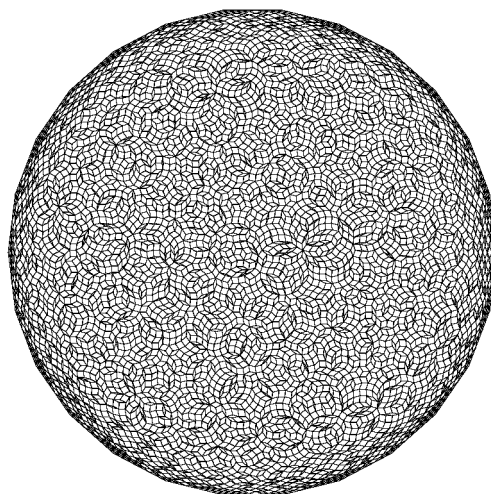


Figure 7: Tiling with 13-fold symmetry obtained by dualization of a grid.

III Summary

The package described in this article contains a collection of introductory *Mathematica* programs to construct planar aperiodic tilings. It is the author's intention to add further programs in the future. Any feedback or suggestions are most welcome.

Bibliography

- Ammann R, Grünbaum B, Shephard GC 1992: Aperiodic tiles, *Discrete Comput. Geom.* **8**, 1–25.
- Baake M 2002: A guide to mathematical quasicrystals, in Suck J-B, Schreiber M, Häussler P (eds.) *Quasicrystals: An Introduction to Structure, Physical Properties, and Applications*, Springer, Berlin, pp 17–48.
- de Bruijn NG 1981: Algebraic theory of Penrose's non-periodic tilings of the plane. I, *Indag. math. (Proc. Kon. Ned. Akad. Wet. Ser. A)* **84**, 39–52; Algebraic theory of Penrose's non-periodic tilings of the plane. II, *Indag. math. (Proc. Kon. Ned. Akad. Wet. Ser. A)* **84**, 53–66.
- Grimm U, Schreiber M 2002: Aperiodic tilings on the computer, in Suck J-B, Schreiber M, Häussler P (eds.) *Quasicrystals: An Introduction to Structure, Physical Properties, and Applications*, Springer, Berlin, pp 49–66; see also <http://mcs.open.ac.uk/ugg2/AperiodicTilings/>.
- Grünbaum B, Shephard GC 1987: *Tilings and Patterns*, W.H. Freeman, New York.
- Janot C 1994: *Quasicrystals: A Primer* (2nd ed.), Clarendon Press, Oxford.
- Maeder R 1990: *Programming in Mathematica*, Addison-Wesley, Redwood City, California
- Penrose R 1974: The rôle of aesthetics in pure and applied mathematical research, *Bull. Inst. Math. Applics. (Southend-on-Sea)* **10**, 266–271.
- Radin C 1999: *Miles of Tiles*, AMS, Providence, Rhode Island.
- Schlottmann M 2000: Generalized model sets and dynamical systems, in Baake M, Moody RV (eds.) *Directions in Mathematical Quasicrystals*, AMS, Providence, Rhode Island, pp 143–159.
- Senechal M 1995: *Quasicrystals and Geometry*, Cambridge University Press, Cambridge.
- Shechtman D, Blech I, Gratias D, Cahn JW 1984: Metallic phase with long-range orientational order and no translational symmetry, *Phys. Rev. Lett.* **53**, 1951–1953.
- Stadnik Z (ed.) 1999: *Physical Properties of Quasicrystals*, Springer, Heidelberg.
- Suck J-B, Schreiber M, Häussler P (eds.) 2002: *Quasicrystals: An Introduction to Structure, Physical Properties, and Applications*, Springer, Berlin.
- Trebin H-R (ed.) 2003: *Quasicrystals: Structure and Physical Properties*, Wiley-VCH, Weinheim

DIMS (Direct-methods program for solving Incommensurate Modulated Structures) on the VEC platform

Hai-fu Fan

Institute of Physics, Chinese Academy of Sciences, Beijing 100080, P. R. China

E-mail: fan@mail.iphy.ac.cn ; WWW: <http://cryst.iphy.ac.cn/>

(Editors' note: the referred to files are included as an Zipped addendum package at the Comp Comm Newsletter No 5 website)

1. Introduction

DIMS is a **D**irect-methods program for solving **I**ncommensurate **M**odulated **S**tructures or, it can also be regarded as a program of **D**irect methods **I**n **M**ultidimensional **S**pace (Fu *et al.*, 1994, 1997; Li *et al.*, 1999). DIMS is based on the multidimensional direct methods developed in our research group in Beijing (Hao *et al.*, 1987; Fan *et al.*, 1993; Sha *et al.*, 1994; Mo *et al.*, 1996) and the Rantan phasing procedure developed in Professor M.M. Woolfson's group in York, England (Yao, 1981). The program is for solving one-dimensionally modulated incommensurate structures and composite structures consists of two subsystems with two axes of the unit cell coincided to each other. For incommensurate modulated structures, DIMS can deal with diffraction data from X-rays, electrons or neutrons, while for composite structures only X-ray diffraction data are considered. There are two versions of DIMS in VEC (Wan *et al.*, 2003). One is merged with other VEC functions, while the other is stand-alone. Both can be invoked on the VEC platform. The former is used for image processing in electron microscopy, while the latter is used for *ab-initio* determination of incommensurate modulated and composite structures. The latter is to be described here. More details of both DIMS and VEC can be found on the web site <http://cryst.iphy.ac.cn/>. Executables of DIMS and VEC and the source codes of DIMS are also available there.

2. Invoking DIMS and preparing the input file

Run the program VEC first, then pull down the menu "Diffraction", select "Ab initio Phasing" and then select "DIMS". This brings up the dialog box "Run DIMS" asking the user to supply a Job file (input file). Click "Browse" to locate the Job file or click "Create" to make a new one (see Fig. 1). The input file is a text file, which can be created either by following the dialog boxes or by using a text editor. A typical input file is shown in Fig. 2. Keywords used in the input file are given in the Appendix, including their definitions and meaning.

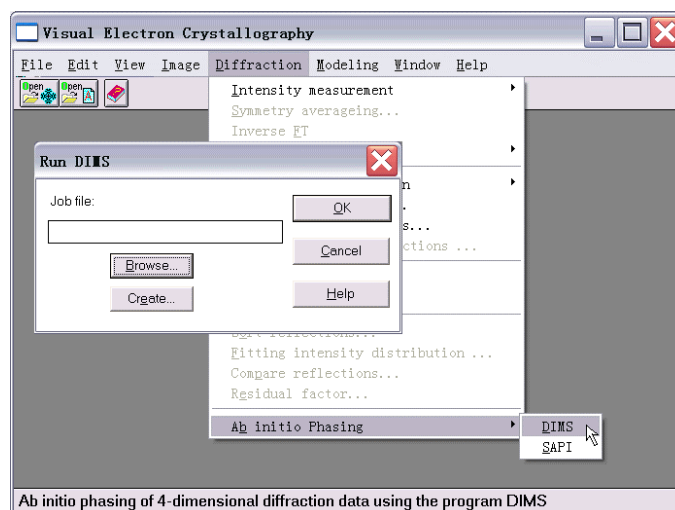


Figure 1.

3. The output files

Three output files are produced by DIMS. They are *.OUT1, *.OUT2 and *.hkml. The first one (*.OUT1) is actually a log file. The second (*.OUT2) is a file containing only experimental structure-factor magnitudes and their phases derived by DIMS. These two files are originally used with the UNIX/DOS version of DIMS. The only output file used on the VEC platform for further calculations is the third one (*.hkml). Once this file has been created by DIMS, it will be opened automatically in graphic mode on the VEC platform in a sub-window (see Fig. 4). Further calculations can then be performed on it. The file can also be opened in text mode on the VEC platform for inspection and editing.

```

g-Na2CO3
C      STATUS      ORDER      PATH      VOID1
C      0           2           3           0
C      NTRIAL      SKIP      VOID2      RANTP
C      20          0           0           1
C      NFS0  NFS1  NFS2  NFS3  NFS4  NFS5  NFS6
C      0      0      0      0      0      0      0
C      ZP0   ZP1   ZP2   ZP3   ZP4   ZP5   ZP6
C      0      0      0      0      0      0      0
C      CLCTR      MAXCL      NCLFIX      RADIUS
C      0.005      1           1           0
C      W1          W2          W3
C      0.20        1.40        1.40
C      MAXREL      KPMIN      KPMAX      PPERC
C      300          1.0        50.0        1.0
C      A1          B1          C1          ALPHA1  BETA1  GAMMA1
C      8.9040      5.2390      6.0420      90.000    101.350  90.000
C      A2          B2          C2          ALFA2    BETA2    GAMA2
C      0.0000      0.0000      0.0000      90.000    90.000    90.000
C      K1          K2          K3
C      0.1820      0.0000      0.3180
C      NOIN      NORMAL  STATIS  BFACTOR  NWLSTEP
C      1          1          1          0.00     16
C      NUMBER      ATOMIC NR  ELEMENT  (CELL CONTENTS)
C      8            11        Na
C      4             6         C
C      12            8         O
C      0             0
C      SPACE-GROUP SYMBOL OR GENERATORS
C      P[C 2/M] -1 S :B
C      DIFFRACTION DATA
C      h      k      l      m      F(obs)      Phase      KN
C      0      0      2      0      61.300      180.000      1
C      0      0      4      0      54.500      0.000       1
C      0      0      5      0      2.400      180.000      1
C      0      0      7      0      2.700      0.000       1
C      0      0      8      0      12.000      0.000       1
C      .      .      .      .      .
C      .      .      .      .      .
C      1      5      0      2      0.100      0.000       0
C      3      5      3      2      0.100      0.000       0
C      7      5      1      2      0.100      0.000       0
C      0      0      0      0     -99.900      0.000       0

```

Figure 2: The input file Na2CO3.key

4. Calculation and display of 4D electron-density maps

Projections and sections of 4-dimensional electron-density maps can be calculated on the VEC platform using the DIMS output file *.hkml.

Example 1.

Using the file Bi-2212.hkml to calculate an $x_3 - x_4$ section $\rho(0.25, 0.00, x_3, x_4)$, the user should first calculate a 3-dimensional hyper-section at $x_1 = 0.25$. Open the *.hkml file in graphic mode or activate the

sub-window containing the file. Then click the button **FT** on the toolbar. This will bring up the dialog box shown in Fig. 3a, choose 'Sections' and set $x_1 = 0.25$. After clicking “OK” another dialog box will appear as is shown in Fig. 3b. Usually there is no need to change any thing in this dialog box; the user can just click “OK” to pass. This will result in a 3-dimensional hyper-section $x_1 = 0.25$ of the 4-dimensional electron-density map. The hyper-section is stored in the disk of the computer. What you see on the screen (the lower-left sub-window in Fig. 4) is only a 2-dimensional section with the first unit cell on the left corresponding to $\rho(0.25, x_2, x_3, 0.00)$. The green lines are unit-cell borders, which are displayed or eliminated by clicking the item “Show/Hide unit-cell border” on the pull-down menu of “Image” (see upper middle of Fig. 4). Unit cells next to the left first one will have different x_4 values according to the 4-dimensional representation of 1-dimensionally incommensurate modulated structures. Operations on this window are not with the 2-dimensional section on the screen but with the 3-dimensional hyper-section in the disk. Now, activate the window and select the item “2d-sections of 4d-Fourier map” on the pull-down menu “Image”. In the pop-up dialog box (lower right of Fig. 4), select the x_3 - x_4 section and set $x_2 = 0$. Then by clicking “OK” the section $\rho(0.25, 0.00, x_3, x_4)$ will be obtained (see the upper-right sub-window in Fig. 4).

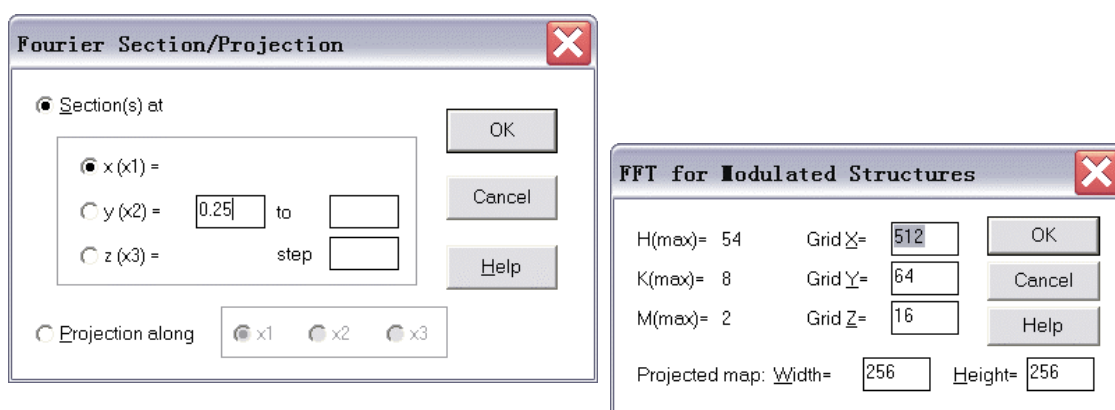


Figure 3:

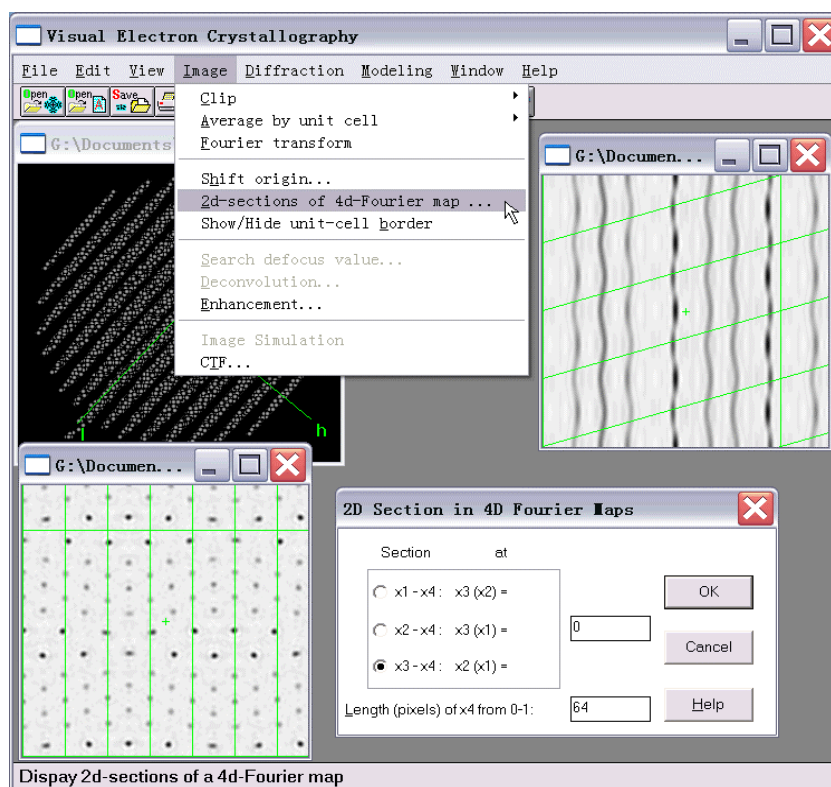



Figure 4:

Example 2.

Using the file PbTiS.hklm to calculate an $x_2 - x_3$ section $\rho(0.25, x_2, x_3, 0.25)$, the user should first calculate a 3-dimensional hyper-section at $x_1 = 0.25$. The operation is the same as that in the previous example. The result is shown in the lower-left sub-window of Fig. 5. Now, pull down the “Image” menu and click on the item “Shift origin” (see upper middle of Fig. 4). In the pop-up dialog box (see the upper-left corner of Fig. 5) set the fractional coordinates $x = 0$, $y = 0$, $z = 0$ and $w = 0.25$. Then the section $\rho(0.25, x_2, x_3, 0.25)$ will come up as is shown in the upper-right sub-window of Fig. 5.

Electron-density maps are displayed by default as half-tone graphs. However they can also be displayed as contour maps. To do this, click on the half-tone graph (lower right of Fig. 6) and then click the button  on the toolbar. Tune the parameters on the pop-up dialog box (lower left of Fig. 6) and click “OK”. The corresponding contoured map will then appear (see upper part of Fig. 6).

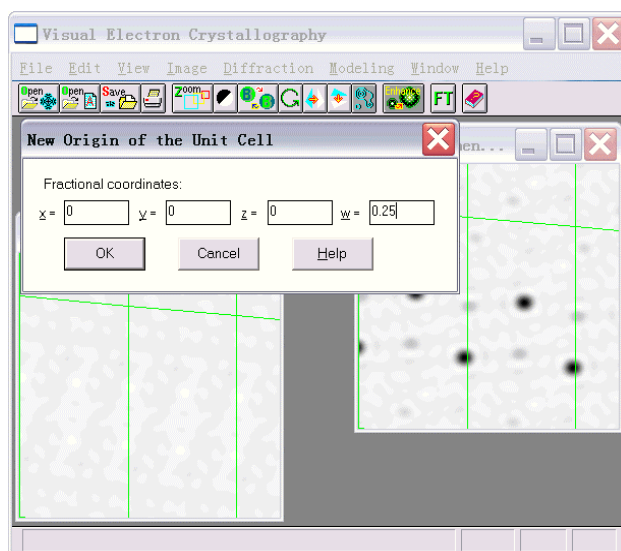


Figure 5:

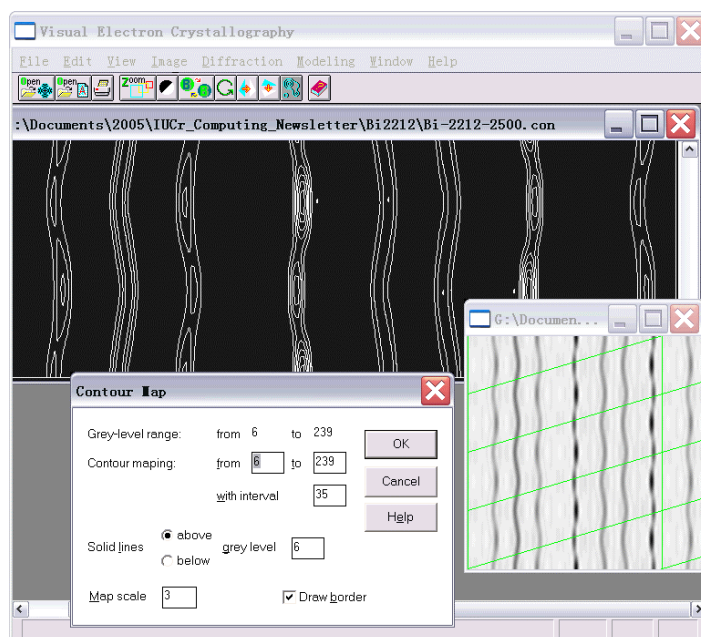


Figure 6:

5. 4D model building

Starting from the *.hklm file, here we use the file **Na2CO3.hklm**, click on the pull-down menu “Modeling” and select “Create Model” (see upper right of Fig. 7). This starts searching for atoms in the 3-dimensional basic structure. When a DOS window appears, press “Enter” to continue. A peak list containing the searching results will appear as shown in the lower part of Fig. 7. The user should assign atom IDs to a set of symmetrically independent peaks. To do this, highlight the peak, click the button “Select” on the peak-list table, then input an atom ID into the pop-up dialog box as shown in the middle of Fig. 7. The atomic ID is the chemical symbol of the element. Optional characters (no spaces) can be added following the symbol. Having finished assigning atom IDs, click the button “Search”. This starts the search of the modulation wave of each atom. Results will be listed in the model file **Na2CO3.mod**, which will be opened automatically and graphically in a sub-window (see the lower part of Fig. 8). The model file can be saved in the disk as a text file; it contains information for starting a least-squares refinement.

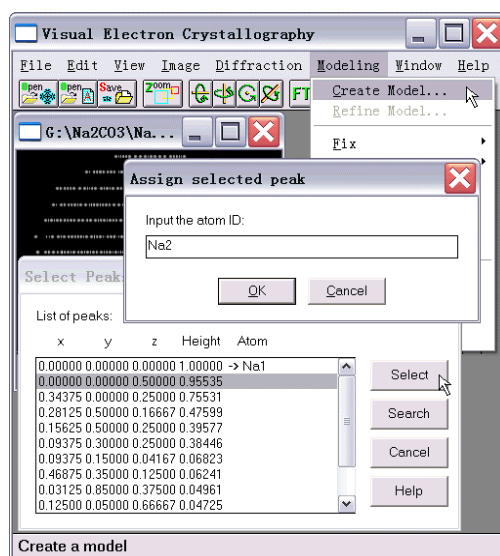


Figure 7:

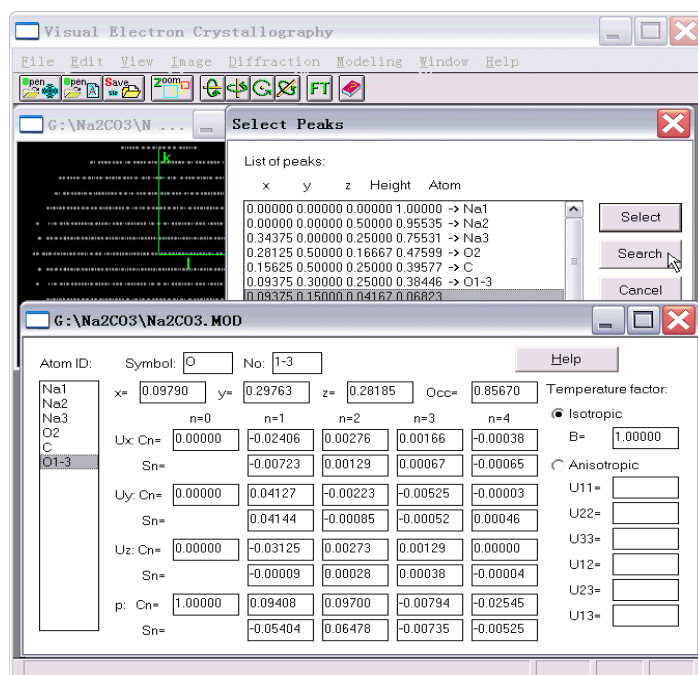


Figure 8:

References

- Fan, H.F., van Smaalen, S., Lam, E.J.W. & Beurskens, P.T. (1993). Direct methods for incommensurate intergrowth compounds I. Determination of the modulation. *Acta Cryst.* **A49**, 704-708.
- Fu, Z.Q. & Fan, H.F. (1994). DIMS --- a direct method program for incommensurate modulated structures. *J. Appl. Cryst.*, **27**, 124-127.
- Fu, Z.Q. & Fan, H.F. (1997). A computer program to derive (3+1)-dimensional symmetry operations from two-line symbols. *J. Appl. Cryst.*, **30**, 73-78.
- Hao, Q., Liu, Y.W. & Fan, H.F. (1987). Direct methods in superspace I. Preliminary theory and test on the determination of incommensurate modulated structures. *Acta Cryst.*, **A43**, 820-824.
- Li, Y., Wan, Z.H. & Fan, H.F. (1999). MIMS — a program for measuring 4-dimensional Fourier maps of incommensurate modulated structures. *J. Appl. Cryst.*, **32**, 1017-1020.
- Mo, Y.D., Fu, Z.Q., Fan, H.F., van Smaalen, S., Lam, E.J.W. & Beurskens, P.T. (1996). Direct methods for incommensurate intergrowth compounds III. Solving the average structure in multidimensional space. *Acta Cryst.* **A52**, 640-644.
- Sha, B.D., Fan, H.F., van Smaalen, S., Lam, E.J.W. & Beurskens, P.T. (1994). Direct methods for incommensurate intergrowth compounds II. Determination of the modulation using only main reflections. *Acta Cryst.* **A50**, 511-515.
- Wan, Z.H., Liu, Y.D., Fu, Z.Q., Li, Y., Cheng, T.Z., Li, F.H. & Fan, H.F. (2003). Visual computing in electron crystallography. *Z. Krist.* **218**, 308-315.
- Yao, J. X. (1981). On the application of phase relationships to complex structures XVIII. RANTAN—Random MULTAN. *Acta Cryst.*, **A37**, 642-644.

Appendix – Keywords for running DIMS

The first line:

A title with no effects to the phasing process

STATUS (default = 0)

0: for unknown structures

1: for known structures, comparison will be made between the phases of satellite reflections input by the user and that derived by DIMS.

PATH (default = 3)

1: for phasing the satellites of incommensurate structures with known phases of the main reflections, the weak-weak relationships are used such that the newly obtained phases of the n^{th} -order satellites are taken as known phases for phasing the $(n+1)^{\text{th}}$ -order satellites. Only one of the 1^{st} -order satellites is assigned a 'known' phase ZP1 to determine the origin of the 4^{th} axis.

2: for phasing the satellites of incommensurate structures with known phases of the main reflections, the weak-weak relationships are neglected for phasing all the satellites. One of each n^{th} -order satellites is assigned a 'known' phase in the phasing procedure. Values of these phase angles are specified under the keyword ZPn.

3: for phasing the satellites of incommensurate structures with known phases of the main reflections, the satellites with order greater than 1 are phased using PATH=2 and then weak-weak relationships are used to determine the origin-dependent phase shift.

4: for phasing composite structures, the weak-weak relationships are neglected.

ZPn

The phase angle of an n^{th} -order satellite, which is used as the origin-fixing reflection for phasing the n^{th} -order satellites.

ORDER (default = 2)

0: phasing for main reflections based on certain known phases of main reflections.

> 0: for PATH = 1, 2 or 3, up to ORDERth-order satellites will be phased with known phases of main reflections.

128: for PATH = 4, only main reflections will be phased.

129: for PATH = 4, all satellites will be phased with known phases of main reflections and, the weak-weak relationships will be neglected.

RANTP (default = 0) active only for acentric space group with PATH = 4

0: random phases of 45/135/225/315 degrees are assigned

1: random phases of 0/180 degrees are assigned

RADIUS (default = 0)

0: input phases in degree.

1: input phases in radius.

MAXREL (default = 300)

Maximum number of Σ_2 relations allowed for a single reflection.

KPMAX (default = 50.0)

Σ_2 -relations with kappa greater than KPMAX will be eliminated.

KPMIN (default = 0.0)

The value of this parameter ranges from 0.0 to 2.0, which is for eliminating Σ_2 -relations with kappa less than KPMIN.

PPERC (default = 1.0)

PPERC \times 100 % reflections (selected from the strongest one downward) will be phased, active only when phasing main reflections of composite structures (PATH=4, ORDER=128).

NTRIAL (default = 50)

Number of trials, i.e. the number of random-starting phase sets (max. NTRIAL = 1024).

SKIP (default = 0)

Skip the first SKIP trials.

NFSn (n = 0, 1, ..., 6)

> 0: output phases will contain up to nth-order satellites, the output phase sets are selected according to the combined figures of merit CFOM.

< 0: The absolute value under the keyword NFSn will be the serial number of the set that you want to output disregarding the value of CFOM.

CLCTR (default = 0.005)

A parameter controlling dynamically the number of cycles of phase iteration

MAXCL (default = 10)

the maximum number of cycles allowed for tangent-formula iteration

NCLFIX (default = 6)

In the first NCLFIX cycles of tangent-formula iteration the known phases are kept fixed, after that they are floatable.

A1, B1, C1, ALPHA1, BETA1, GAMMA1

Unit-cell parameters of the basic structure of the incommensurate modulated structure, or of the first subsystem of the composite structure.

A2, B2, C2, ALPHA2, BETA2, GAMMA2

Unit-cell parameters of the second subsystem of the composite structure.

K1, K2, K3

The a^* , b^* and c^* components of the modulation wave vector

$$\mathbf{q} = k_1 \mathbf{a}^* + k_2 \mathbf{b}^* + k_3 \mathbf{c}^*$$

W1 (default = 0.2), W2 (default = 1.4), W3 (default = 1.4)

Weights of the figures of merit ABSFOM, PSI-ZERO and RISIDUAL in the calculation of the combined figure of merit CFOM

NOIN

In the cell contents, the top NOIN chemical elements belong to the first subsystem of the composite structures, active only when phasing main reflections of composite structures (PATH=4 and ORDER=128).

NORMAL (= 0 or 1)

Indicates one of the two strategies for scaling Fobs, active only when PATH=4 and ORDER=128.

STATIS

0: no WILSON statistics will be performed

1: WILSON method is used to scale Fobs

2: K-curve method is used to scale Fobs, active only when PATH=4 and ORDER=128.

BFACTOR

0.0: the B-factor from WILSON statistics is used for scaling, else: BFACTOR is used instead of the B-factor from WILSON statistics; active only when PATH=4, ORDER=128 and STATIS=1.

NWLSTEP (default = 16)

For the WILSON statistics, the reciprocal space will be divided in NWLSTEP zones.

ELEMENT

Chemical symbol of atoms in the unit cell.

ATOMIC NR

Atomic number of the specified chemical element

NUMBER

Number of atoms in the cell

SUPERSPACE GROUP: TWO-LINE SYMBOL or GENERATORS

The superspace-group symmetry is expressed either by a two-line symbol or a set of generators.

As an example the two-line symbol for the incommensurate modulated structure of γ -Na₂CO₃ is

$$P[C\ 2/M]-1\ S :B$$

For more details, the user is referred to Fu Zheng-qing & Fan Hai-fu (1997) "A computer program to derive (3+1)-dimensional symmetry operations from two-line symbols" *J. Appl. Cryst.* **30**, 73-78.

If generators are to be used, the user should first specify the number of generators before listing the elements of the first generator. Each generator should be ended with a blank line. As an example the generators for γ -Na₂CO₃ not including operations of the centered lattice are expressed as

2					
-1	0	0	0	0.0000	
0	1	0	0	0.0000	
0	0	-1	0	0.0000	
0	0	0	-1	0.5000	
1	0	0	0	0.0000	
0	-1	0	0	0.0000	
0	0	1	0	0.0000	
0	0	0	1	0.5000	

KN

Indicates the preceding phase is known or not

- 0: unknown, its value is to be derived, the listed value will NOT take part in the derivation, however in the case of STATUS = 1 the listed phases will be compared with that derived from DIMS.
- 1: known, it will be used as starting phase to derive unknown phases.
- 2: a random phase will be assigned

MK

Indicates whether the reflection will be rejected in the phasing process.

- 1: rejected, a random phase will be given to this reflection in the output file.
- 1: not rejected.

DN

For STATUS=1, indicates the difference between the given and the derived phases.

DIMS (Direct-method program of solving Incommensurate Modulated Structures)/VEC applications

Hai-fu Fan

Institute of Physics, Chinese Academy of Sciences, Beijing 100080, P. R. China

E-mail: fan@mail.iphy.ac.cn ; WWW: <http://cryst.iphy.ac.cn/>

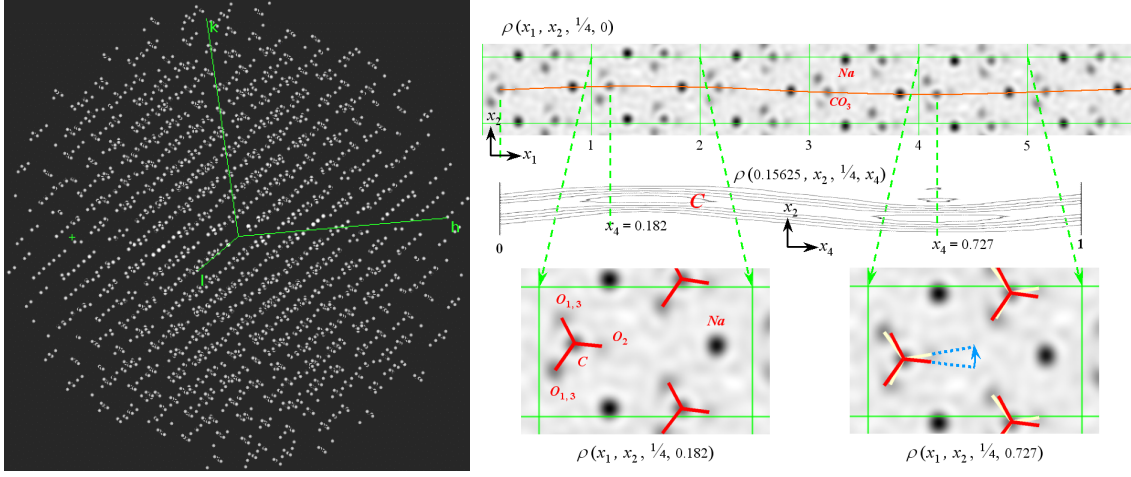
(Editors' note: the referred to files are included as an Zipped addendum package at the Comp Comm Newsletter No 5 website)

1. Introduction

The integration of DIMS (Fu et al., 1994, 1997; Li et al., 1999) and VEC (Wan et al., 2003) provides an intuitive and automatic way of solving incommensurate structures. Calculations are visually performed on the VEC platform mostly via mouse clicks. Examples are given here involving two incommensurate modulated structures and the 4-dimensional basic structure of a composite crystal. As will be seen, structural details including the incommensurate modulation can be observed objectively prior to the model building and least-squares refinement. For detailed operations of the program DIMS/VEC the reader is referred to the paper "DIMS on the VEC platform" in this newsletter.

2. Direct observation of the incommensurate modulation of $\gamma\text{-Na}_2\text{CO}_3$

Crystals of $\gamma\text{-Na}_2\text{CO}_3$ have a one-dimensionally modulated incommensurate structure with unit cell parameters of the basic structure $a = 8.904$, $b = 5.239$, $c = 6.042\text{\AA}$, $\beta = 101.35^\circ$ and the modulation wave vector $\mathbf{q} = 0.182\mathbf{a}^* + 0.318\mathbf{c}^*$. The superspace group is $P[C\ 2/m] -1\ s$ (two-line symbol used in DIMS). Van Aalst et al. (1976) originally solved the modulated structure by trial-and-error method. Hao et al. (1987) used their data to test the multidimensional direct method. 300 largest main reflections, 250 largest first order satellites and 150 largest second order satellites from the experimental data were selected for the test. The program SAPI (Yao et al., 1985) was used to derive phases of main reflections, based on which the multidimensional direct method was used to phase satellite reflections. There is no need to know the basic structure in advance. In the present test, the input file **Na2CO3.key** was constructed with the same data used by Hao et al. (1987). The output file **Na2CO3.hklm** was produced and opened in a sub-window on the VEC platform (see Fig. 1). The file contains the original input data together with the direct-method phases of satellite reflections. Fig. 2 shows sections of the 4-dimensional electron density map of $\gamma\text{-Na}_2\text{CO}_3$ calculated with the file **Na2CO3.hklm**. The top row of Fig. 2 shows the half-tone graphic section of the 4-dimensional electron-density map at $x_3(z) = 1/4$. Six unit cells are plotted along the \mathbf{x}_1 axis. Since the modulation wave vector \mathbf{q} has a component $q_1 = 0.182$ along \mathbf{a}^* , the modulation period should be about 5.5 unit cells along the \mathbf{x}_1 axis. Consequently the first unit cell on the left of the top row corresponds to $x_4 = 0.0$, while the sixth unit cell corresponds to $x_4 \approx 1.0$. This is evident comparing the top row and the middle row, the contoured section $\rho(0.15625, x_2, 1/4, x_4)$. As is seen the top section reveals clearly sodium atoms and CO_3 groups. It is seen also that the x_2 coordinate of carbon atoms varies along the \mathbf{x}_1 axis from one unit cell to the other indicating the positional modulation of the carbon atoms as shown by the red curve. This can be seen more precisely on the section $\rho(0.15625, x_2, 1/4, x_4)$. By comparing the second and the fifth unit cell (see the bottom row of Fig. 2), it is observed that the modulation of the CO_3 groups performs an anticlockwise rotation around the axis through the carbon atom perpendicular to the (\mathbf{a}, \mathbf{b}) plane (\mathbf{a} and \mathbf{b} are respectively the projection of \mathbf{x}_1 and \mathbf{x}_2 along the direction perpendicular to the 3-dimensional physical space). All these features are consistent with the original result of Van Aalst et al. (1976). One of the most important differences between DIMS/VEC and the trial-and-error method is that all features of the structural modulation of $\gamma\text{-Na}_2\text{CO}_3$ are visualized on the direct-method phased electron-density map, which does not rely on any assumptions concerning the form of modulation waves and is obtainable prior to model building and structure refinement.



Figures 1 (left) and 2 (right)

3. Visualizing structural modulation of Bi-2212

The incommensurate structure of the high-Tc superconductor $\text{Bi}_2\text{Sr}_2\text{CaCu}_2\text{O}_y$ (Bi-2212) has been extensively studied in a number of laboratories over the world. However the results are not completely consistent with each other. Here DIMS/VEC produces the visualized structural modulation without relying on any assumptions on the modulation waves. The data used in this test is the same as that of Fu et al. (1995). Crystals of Bi-2212 belong to the superspace group $N[\text{Bbmb}]1-11$ (two-line symbol used in DIMS) with unit cell parameters of the basic structure $a = 5.422$, $b = 5.437$, $c = 30.537\text{\AA}$ and the modulation wave vector $\mathbf{q} = 0.22\mathbf{b}^* + \mathbf{c}^*$. SAPI was used to derive phases of the main reflections.

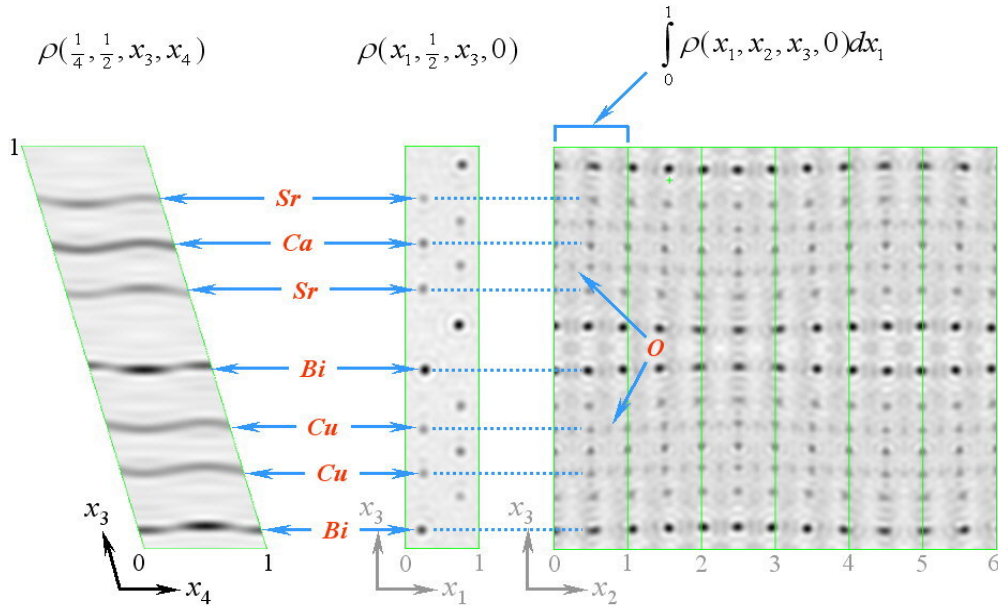


Figure 3:

Fourier recycling was used to determine the basic structure. The input file **Bi-2212.key** was then constructed, with which DIMS/VEC produced the output file **Bi-2212.hklm**. Electron-density maps were calculated on the VEC platform. The 4-dimensional electron-density function of Bi-2212 projected along the x_1 axis is shown on the right of Fig. 3 giving an overview of the incommensurate structure. Six unit cells are plotted along the x_2 axis. All metal atoms and the oxygen atoms on Cu-O layers are clearly seen. The section at $x_2 = 1/2$, $x_4 = 0$ shown in the middle of Fig. 3 contains all the independent metal atoms.

Their modulation is shown on the section $\rho(\frac{1}{4}, \frac{1}{2}, x_3, x_4)$ (on the left of Fig. 3). Fig. 4 shows atoms on the Cu-O layer and the modulation of the oxygen atom O(1). Fig. 5 shows the saw-tooth modulation of the oxygen atom O(4). It should be emphasized that the saw-tooth modulation here is not a result of least-squares refinement based on a guessed model. In contrast it is revealed objectively before any efforts of model building.

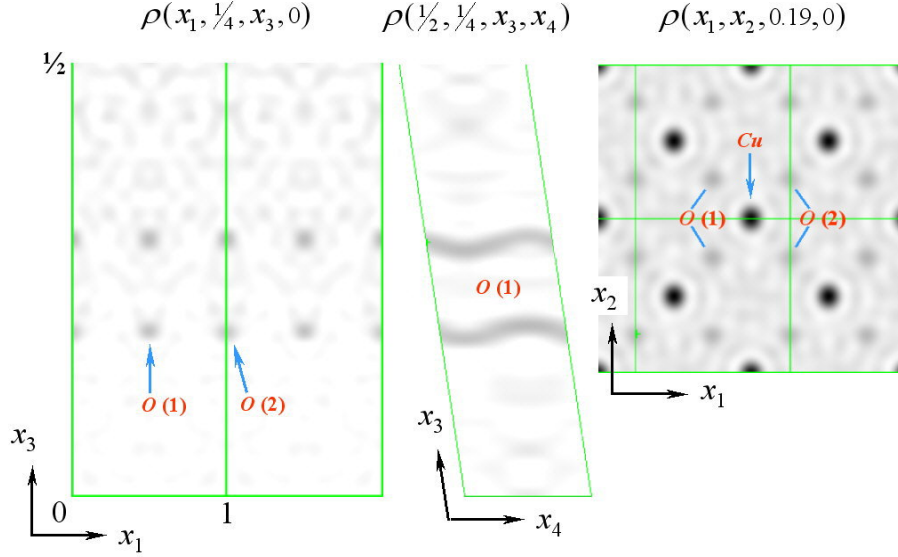


Figure 4:

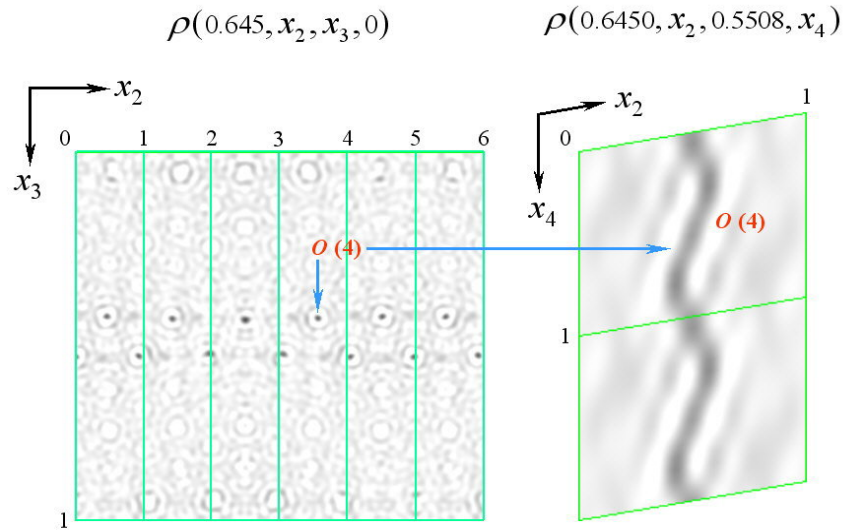
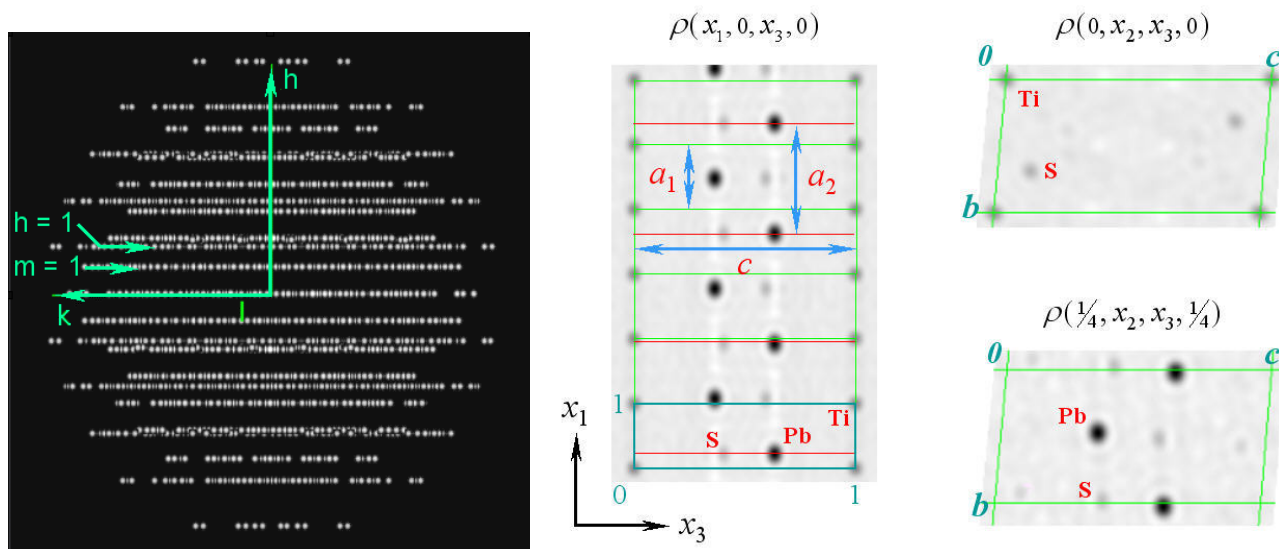


Figure 5:

4. Solving the 4-dimensional basic structure of the composite crystal $(\text{PbS})_{1.18}\text{TiS}_2$

The composite structure of $(\text{PbS})_{1.18}\text{TiS}_2$ (Van Smaalen et al., 1991) belongs to the space group $C2/m(\alpha, 0, 0)$ s-1. It consists of two subsystems: the subsystem TiS_2 with $a_1 = 3.409$, $b_1 = 5.880$, $c_1 = 11.760\text{\AA}$, $\alpha_1 = 95.29^\circ$ and the subsystem PbS with $a_2 = 5.800$, $b_2 = 5.881$, $c_2 = 11.759\text{\AA}$, $\alpha_2 = 95.27^\circ$. Within the experimental error we have $b_1 = b_2$, $c_1 = c_2$, and $\alpha_1 = \alpha_2$. Unlike conventional incommensurate modulated structures, there are no 3-dimensional basic structures corresponding to a composite structure. For $(\text{PbS})_{1.18}\text{TiS}_2$, the basic structure is a 4-dimensional one. It is more complicated to determine such a basic structure than to fine the modulation of $(\text{PbS})_{1.18}\text{TiS}_2$, since there are no known phases available for

the direct-method phasing except the origin and enantiomorph fixing ones. The following test shows that DIMS/VEC is capable of solving the 4-dimensional basic structure in a straightforward manner. The input file **PbTiS.key** contains only main reflections. The symmetry is assumed to be non-centrosymmetric. The output file from DIMS/VEC is **PbTiS.hklm**, which is opened in a sub-window as shown in Fig. 6. Sections of the 4-dimensional electron-density maps calculated from **PbTiS.hklm** are shown below. On the left of Fig. 7 we see the “chimney and ladder” structure along the x_1 axis constructed by the TiS_2 subsystem with the period a_1 and the PbS subsystem with the period a_2 . On the right of Fig. 7 there are sections through the TiS_2 layer and PbS layer parallel to the (b, c) plane. Note that a_1 , a_2 , b and c are respectively the projection of the axes x_1 , x_4 , x_2 and x_3 along the direction perpendicular to the 3-dimensional physical space. Again all the structural features are visible on the direct-method phased electron-density map before any efforts of model building and structure refinement.



Figures 6 (left) and 7 (right)

References

- Fu, Z.Q. & Fan, H.F. (1994). DIMS --- a direct method program for incommensurate modulated structures. *J. Appl. Cryst.*, **27**, 124-127.
- Fu, Z.Q., Li, Y., Cheng, T.Z., Zhang, Y.H., Gu, B.L. & Fan, H.F. (1995). Incommensurate modulations in Bi-2212 high-Tc superconductor revealed by single-crystal X-ray analysis using direct methods. *Science in China*, **A38**, 210-216.
- Fu, Z.Q. & Fan, H.F. (1997). A computer program to derive (3+1)-dimensional symmetry operations from two-line symbols. *J. Appl. Cryst.*, **30**, 73-78.
- Hao, Q., Liu, Y.W. & Fan, H.F. (1987). Direct methods in superspace I. Preliminary theory and test on the determination of incommensurate modulated structures. *Acta Cryst.* **A43**, 820-824.
- Li, Y., Wan, Z.H. & Fan, H.F. (1999). MIMS — a program for measuring 4-dimensional Fourier maps of incommensurate modulated structures. *J. Appl. Cryst.*, **32**, 1017-1020.
- Van Aalst, W., Den Hollander, J., Peterse, W.J.A.M. & De Wolff, P.M. (1976). The modulated structure of γ - Na_2CO_3 in a harmonic approximation. *Acta Cryst.* **B32**, 47-58.
- Van Smaalen, S., Meetsma, A., Wiegers, G. A. and De Boer, J. L. (1991). Determination of the modulated structure of the inorganic misfit layer compound $(\text{PbS})_{1.18}\text{TiS}_2$. *Acta Cryst.*, **B47**, 314-325.
- Wan, Z.H., Liu, Y.D., Fu, Z.Q., Li, Y., Cheng, T.Z., Li, F.H. & Fan, H.F. (2003). Visual computing in electron crystallography. *Z. Krist.* **218**, 308-315.
- Yao, J.X., Zheng, C.D., Qian, J.Z., Han F.S., Gu, Y.X. & Fan, H.F. (1985). SAPI-85: a computer program for automatic solution of crystal structures from X-ray diffraction data. Institute of Physics, Chinese Academy of Sciences, Beijing 100080, P.R. China.

Collection and visualization of single crystal data of incommensurate crystals.

Rob W.W. Hoof

Bruker AXS BV, Delft, The Netherlands; Email: rob.hoof@bruker-axs.nl ; WWW: <http://www.bruker-axs.nl/>

Introduction

With the change from point detectors to solid state 2D detectors, all “small molecule” single crystal measurement techniques have undergone large changes. Researchers interested in incommensurately modulated or composite structures have been more reluctant to move away from using point detectors. This paper tries to find out why, and explains what the problems of the modern equipment for such samples can be. We will also explore what a 2D detector can do for these samples.

Point detector systems

Measuring an incommensurate crystal with a point detector was difficult. The software for such systems used a three dimensional lattice description, and any higher-dimensional description required extra work. Many laboratories specializing in these samples made their own adaptations to the data collection software.

Unit cell determination

The difficulties with incommensurate structures arise already at the determination of the unit cell. There are basically two different cases that can occur: (1) if the satellite reflections are weak, it is possible that none are found during the initial search. Indexing procedures will then find the basic lattice, but without a proper X-ray photo (on film) to visualize reciprocal space it is impossible to see that the sample shows incommensurate diffraction. (2) satellite peaks are found in the initial search, hence not all reflections can be indexed to a three-dimensional lattice. Sometimes (for modulated structures, but not for composites) it can help to index on the strongest reflections only. Specialized indexing software like *dirax* [1] has been capable of finding the basis cell for such samples based on the complete reflection list, and recent versions of *dirax* can also identify a single modulation vector from the reflections that do not fit the 3D lattice. Only recently, Pilz *et al.* [2] describe an indexing method tailor made for incommensurate crystals.

Data collection

Data collection on incommensurate crystals on point detector systems also needed special attention. Special versions of the data collection software would be used to collect first the main lattice and then the satellites; alternatively a commensurate super cell could be defined with appropriate absents conditions.

Visualization

True computational visualization of the data collected with a point detector is not very useful for determining the background of a modulated structure. The normal way to visualize the data on such a system was to make a Weissenberg or precession picture, or a rotation photo (on film) of the unaligned or the aligned crystal on a CAD4.

CCD Detector systems

A CCD detector is an integrating detector with a good spatial resolution. The integration is over time, and hence over the rotation of the crystal. Spot localization and separation within the image is excellent, but between frames this is dependent on the rotation angle per frame.

This is exactly reversed for a point detector system, where the hardware integrates all photons coming in through the entire input window, but localization in time for the reflections is very good.

These differences between the systems have their implications on data processing, but it is important to realize that both point-detector and 2D detector perform integration in the hardware, so that this is not a new phenomenon.

Unit cell determination

Cell determination with a CCD detector system is in principle much easier than using the point detector system since there are a lot more reflections available for indexing than are generally searched for with a point detector. Furthermore, reciprocal space is sampled more systematically using a 2D detector. Unfortunately, however, the accuracy of the locations of the reflections in three-dimensional reciprocal space determined from wide-angle rotation images is not good: the positional accuracy in the two dimensions on the detector are excellent, but the accuracy in the rotation direction is relatively poor. Especially for incommensurates (as well as for twinned crystals) this can pose a serious complication.

One possible solution to this is so-called *fine slicing*, whereby many images are made, each of which is only a small rotation (e.g. 0.3 degrees). Each reflection should be seen in at least three subsequent diffraction images, such that the accurate centroid can be determined by interpolation.

An alternative method to get accurate 3-dimensional reflection positions is the phi/chi [3] procedure whereby each reflection is scanned twice using different scans (reminiscent of the SETANG procedure on the CAD4 diffractometer). The accurate three-dimensional location of the reflection is then calculated from the intersection of the two different lines through reciprocal space. This procedure requires that the goniostat can scan multiple axes synchronously.

If the satellites are relatively weak, they may pass unnoticed with any indexing procedure, especially since images taken with the purpose of indexing do not normally have long exposure times. If this is the case, most likely these spots will show up as unexplained effects on the actual diffraction images used for the data collection; enough for an experienced crystallographer to pick this up afterwards. Fortunately there is in most cases a second chance to find the correct cell using the images from the data collection.

Data collection

The data collection on an incommensurate crystal with a 2D detector system is automatically right: In first instance, data collection strategies are based on “sweeping” the asymmetric part of reciprocal space, and this is only dependent on the lattice symmetry and the orientation of the reciprocal axes in space, and not on the actual length of the axes. Since the incommensurate vector does not affect the lattice symmetry, the same strategy determination can be used. Even if the sample is not known to be incommensurate, any strategy designed for the main lattice will also collect all relevant satellite reflections.

So far for the simple bit. There are two more aspects of the data collection that need extra attention.

1. ***Spatial resolution.*** The spatial resolution of a measurement is influenced by the beam divergence (optics), the crystal, the point spread of the detector and the geometry of the measurement. The

distance between the crystal and the detector must be chosen large enough to be able to separate the main lattice reflections and the satellites. A good separation of spots is especially important as the main lattice reflections can be orders of magnitude stronger than the satellites. Earlier generations of CCD cameras are especially sensitive to this dynamic range, as very strong reflections could cause *blooming* effects that mask out nearby weak satellites. The latest generation of CCD chips now have anti-blooming features that do not decrease the sensitivity of the detector; this is an advantage for modulated samples.

Since the satellites are weak, it is tempting to use modern X-ray optics to increase the intensity of the beam. However, any optic that adds intensity necessarily also adds divergence to the beam; this makes it potentially more difficult to separate the closely spaced diffraction spots. The usability of optics which add so much divergence that a larger crystal-to-detector distance is required is doubtful.

2. **Data collection time.** One must make sure that both satellites and main lattice reflections can be integrated accurately as there are limitations in the dynamic range of the detector and the analog to digital conversion. The conversion is normally done with a 16 bit ADC, but in practice, the dynamic range that can be caught in one exposure is less than 65536:1, more like 10000:1. The dynamic range of a measurement can be increased by repeating all measurements at different exposure times, or, saving a bit of measurement time, by making faster exposures only for the frames that contain overflowed pixels. The anti-blooming features mentioned earlier can prevent negative effects of the overflowed reflections on neighboring satellites. Reflections with intermediate intensities which can be integrated accurately in both long and short exposure time measurements can be used to verify the scaling between the repeated measurements.

Lack of experience with the spatial resolution and dynamic range for 2D detector systems have withheld scientists specialized in incommensurate structures from switching their point detector systems to 2D detector systems. Now that this experience is accumulating and further improvements to the systems have been made, it can be shown that the quality of the data obtained from a CCD can be superior to point detector data even though the latter are *photon counted*.

The fact that on a 2D detector data are not collected reflection-by-reflection, but in sweeps through reciprocal space brings with it that many reflections will be measured more than once. In fact, this redundancy is the proper way to increase data quality when using an instrument with a 2D detector. The best results are obtained when redundant measurements differ in as many parameters as possible. This type of true redundancy is best obtained by using a 4-axis goniometer. True redundancy reduces the effects of systematic errors in the measurement and provides more detail to absorption correction procedures.

Visualization

Visualization from 2D diffraction data can currently be done in two ways:

- A peak search is performed over (part of) the collected data. The location of each of the peaks is shown in a 3D model that can be manipulated in real time to visualize the unit cell in reciprocal space. Tools for measuring distances between rows and planes of reflections can help to determine the modulation vectors. This procedure can also be very instructive for twinned or fragmented crystals.
- A set of “synthesized precession images” is created. These images are a projection of 3D reciprocal space onto a plane. Each pixel in all frames is assigned a reciprocal space coordinate, and mapped to the synthesized image. Because of this procedure, not only Bragg peaks, but also diffuse scattering effects will show up in the synthesized precession images.

During the synthesis, the intensity of all measured pixels mapping to a single pixel in the precession image is averaged. This averaging takes care of large differences in Lorentz factor, but this does not make the synthesized image quantitatively correct.

For visualization of incommensurate structures, it is possible to calculate a pseudo-precession image for a plane that has both main lattice and satellite reflections. It is also possible to make an image of a plane that only contains satellite reflections.

In general, this technique is very good to help understanding any effects in reciprocal space that are much more difficult to grasp from the rotation images directly.

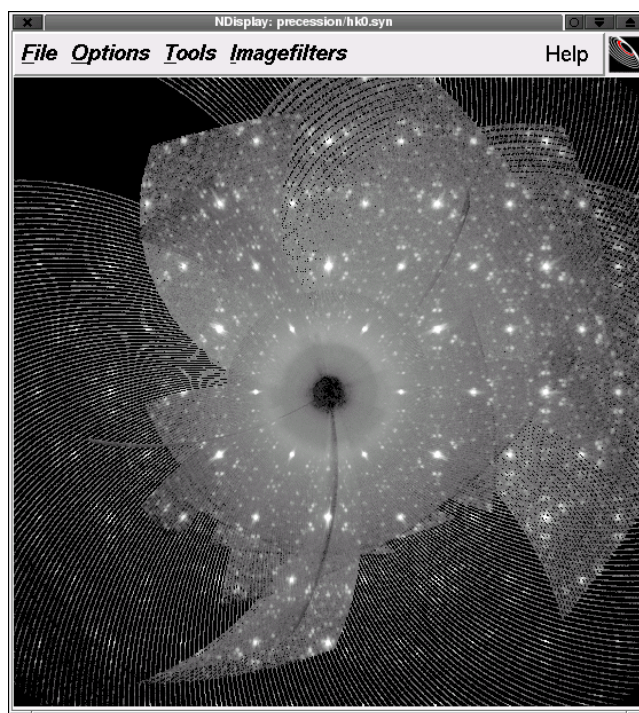


Fig 1: Synthesized *hk0* precession image of an interesting hexagonal sample.

Conclusion

The objections against switching from point detectors to 2D detectors for incommensurate structures are no longer valid. The advantages of *speed* and *true redundancy* for the new instruments result in better quality data. The *improved visualization* tools make additional x-ray film camera's and crystal alignment superfluous.

Acknowledgments

The author thanks his colleagues Eric Hovestreydt, Frank van Meurs, Michael Ruf and Leo Straver for critical examination of the text.

References

- [1] A.J.M. Duisenberg, *Indexing in single-crystal diffractometry with an obstinate list of reflections*, J. Appl. Cryst. (1992), **25**, 92-96
- [2] K. Pilz, M. Estermann and S. van Smaalen, *Automatic indexing of area-detector data of periodic and aperiodic crystals*, J. Appl. Cryst. (2002), **35**, 253-260
- [3] A.J.M. Duisenberg, R.W.W. Hooft, A.M.M. Schreurs and J. Kroon, *Accurate cells from area-detector images*, J. Appl. Cryst. (2000), **33**, 893-898

Visualization and Analysis of Single Crystal Time-of-Flight Neutron Scattering Data using ISAW

Dennis Mikkelsen^a, Arthur J. Schultz^b, Ruth Mikkelsen^a, Thomas Worlton^b

^aUniversity of Wisconsin-Stout, Menomonie, WI, USA and ^bArgonne National Laboratory, Argonne, IL, USA; Email: mikkelsond@uwstout.edu, mikkelsonr@uwstout.edu, ajschultz@anl.gov, tgworlton@anl.gov; WWW: <http://www.pns.anl.gov/computing/isaw/>

Abstract

Single crystal time-of-flight neutron scattering experiments provide information about three dimensional regions in reciprocal space. New software to visualize and analyze such data has recently been developed in the context of the Integrated Spectral Analysis Workbench at the Intense Pulsed Neutron Source division of Argonne National Laboratory. This software is user-friendly, highly interactive, and includes a novel 3D view of reciprocal space, that has been useful when dealing with twinned or multiple crystals.

Introduction

The Integrated Spectral Analysis Workbench (ISAW) is a large collection of software objects for neutron scattering data access, visualization and analysis. ISAW has been developed over the last six years by a team from the Intense Pulsed Neutron Source (IPNS) division of Argonne National Laboratory and the University of Wisconsin-Stout, with support from the National Science Foundation. ISAW is implemented in JAVA for portability and is freely available under the GNU GPL[1].

ISAW has support for several instrument types, including single crystal diffractometers, and is being extended to more instrument types. After a brief overview of the structure of the software, some of the major features of ISAW that support single crystal diffraction data will be described in more detail.

ISAW Overview

ISAW is built around several fundamental concepts. Internally, raw and partially reduced data are stored in "DataSet" objects that are collections of "Data" blocks (spectra). The DataSets hold the raw data along with meta-data needed for data analysis, such as detector positions, initial flight path length, sample orientation, etc. Raw data is loaded into DataSets by data "Retrievers". Data can be "retrieved" from IPNS run files, NeXus files, a remote file server, etc. Individual DataSets can be viewed in various ways by DataSet viewers. Data analysis steps are implemented in self-describing "Operator" objects. A typical analysis sequence can be carried out manually by loading data, and applying analysis and visualization operations using the main ISAW GUI. Alternatively, the sequence of steps can be controlled by a script written in ISAW's scripting language or in Jython. Common sequences of operations can also be easily combined into a "Wizard" consisting of a sequence of forms representing the steps of the peak indexing and integration process. Data can also be written out in NeXus format. Figure 1 is a screen dump showing the ISAW control panel and two views of data from the single crystal diffractometer at IPNS.

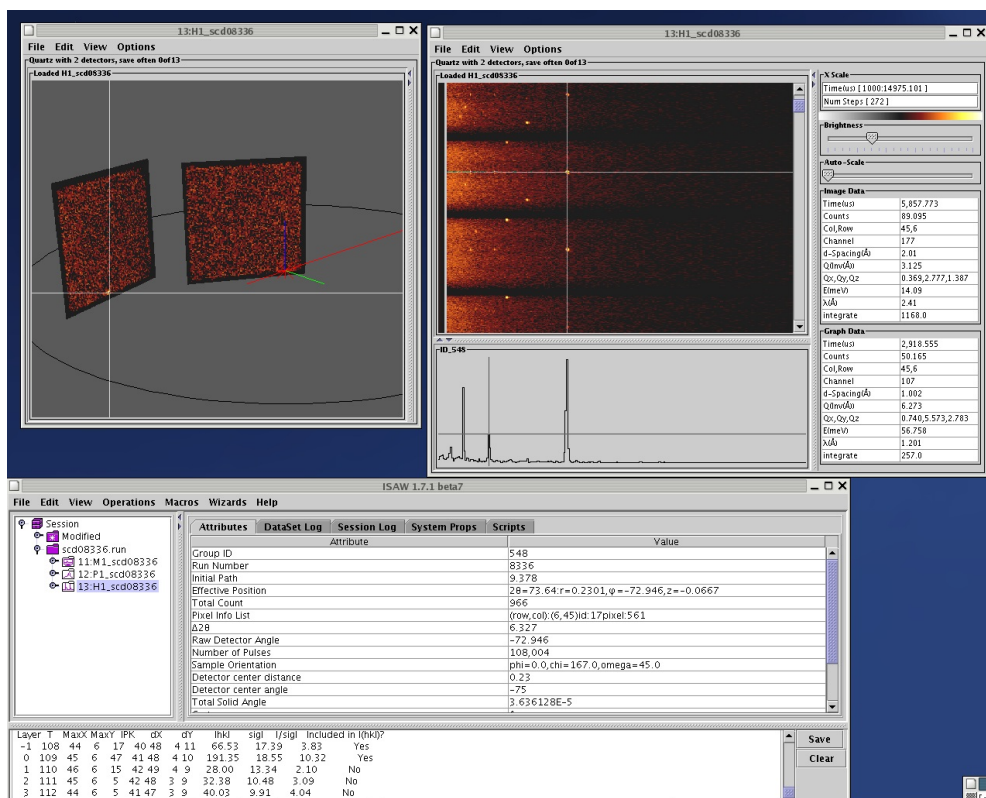


Figure 1

ISAW SCD Support

The SCD at IPNS uses time-of-flight neutron scattering measurements to provide information about three dimensional regions in reciprocal space. The SCD was recently upgraded to employ two area detectors, so that a larger volume of reciprocal space can be measured simultaneously. Since a new data acquisition system was needed and some of the legacy software was written assuming only one area detector located at 90 degrees, it was decided to build new software to support single crystal diffraction in the context of ISAW.

In principle, providing support for a new instrument type in ISAW merely requires providing a suitable set of operator objects to carry out the required data analysis steps. In practice, it has also been helpful to provide some customized viewers for the data from a newly supported instrument type and to provide some user friendly framework tuned for that instrument type. Finally, specialized software components for tasks such as instrument calibration may also be needed.

In the case of single crystal diffractometers, operators to carry out essential analysis steps such as finding, indexing and integrating peaks in the 3D volume data were implemented. In addition, user-friendly "Wizards" were implemented to organize the steps and guide the user through the data analysis process. Figure 2 shows a form from one of the Wizards.

File View Project Directory Help

Form 4: IntegrateMultiRunsForm

CONSTANT PARAMETERS

Raw Data Path	/usr2/SCD_TEST /	Browse	<input checked="" type="checkbox"/>
Peaks File Output Path	/usr2/SCD_TEST /	Browse	<input checked="" type="checkbox"/>
Experiment name	quartz		<input checked="" type="checkbox"/>
SCD Calibration File	/usr2/SCD_TEST /instprm.dat	Browse	<input checked="" type="checkbox"/>
SCD Calibration File Line to Use	-1		<input checked="" type="checkbox"/>

USER SPECIFIED PARAMETERS

Run Numbers	08336:08337	<input type="checkbox"/>
Centering Type	primitive	<input type="checkbox"/>
Time-Slice Range	-1:3	<input checked="" type="checkbox"/>
Amount to Increase Slice Size By	1	<input checked="" type="checkbox"/>
Append to File?	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Integrate 1 peak method	MaxItoSigI	
Box Delta x (col) Range	-2:2	<input checked="" type="checkbox"/>
Box Delta y (row) Range	-2:2	<input checked="" type="checkbox"/>

RESULTS

Integrated Peaks File	/usr2/SCD_TEST /quartz.integrate	Browse	<input type="checkbox"/>
-----------------------	----------------------------------	--------	--------------------------

Reset IntegrateMultiRunsForm Progress Do

Reset All Wizard Progress: 3 of 4 Forms done Do All

First Form Back One Forward One Last Form

Figure 2

Interactive viewers to display a 3D view of reciprocal space as well as arbitrary slices through reciprocal space were designed and implemented, as shown in figures 3 and 4.

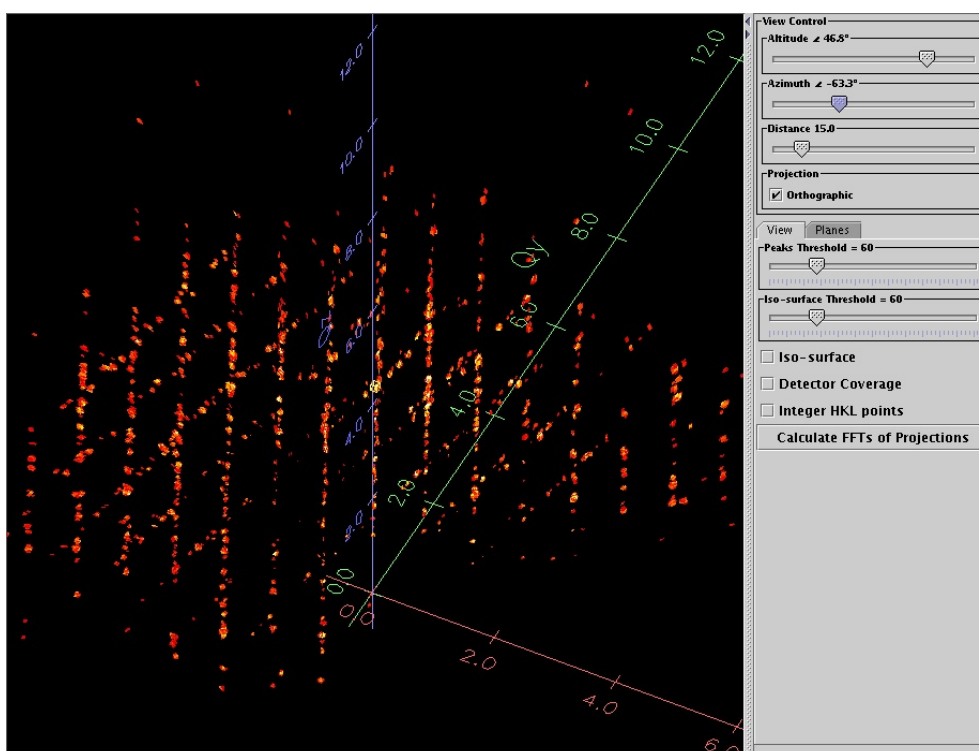


Figure 3

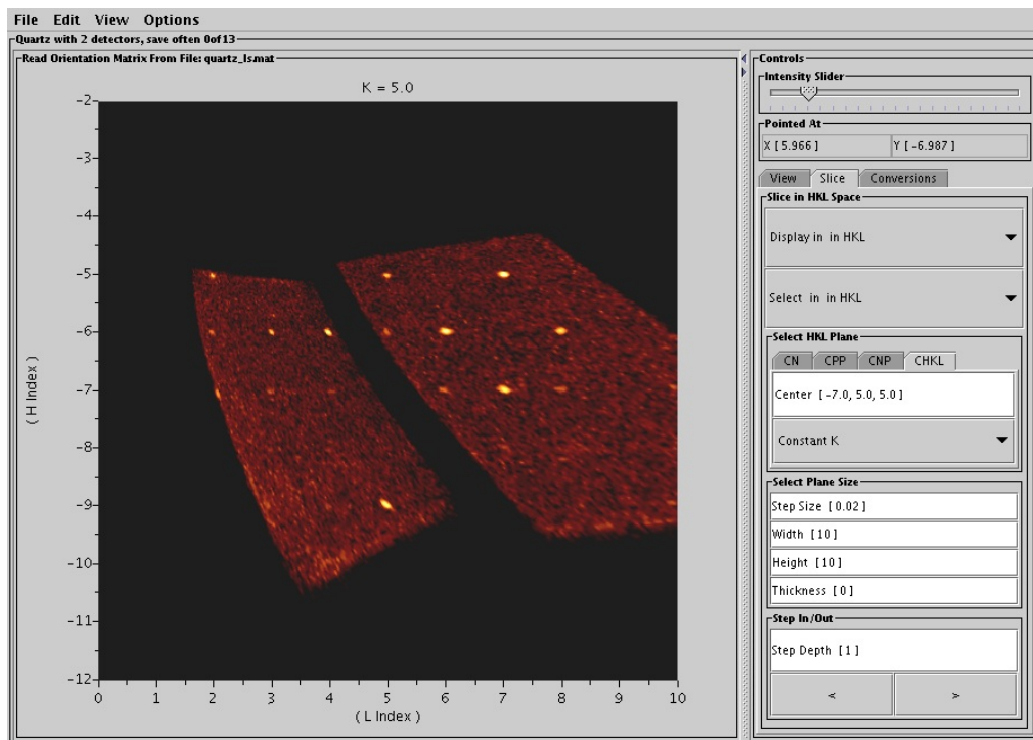


Figure 4

Finally, a new calibration system was implemented to adjust instrument parameters such as the nominal detector positions, orientations and sizes. The calibration operator uses the known lattice parameters of a calibration sample such as quartz, and a Marquardt type optimization routine[4] to adjust instrument parameters to minimize the sum squared differences between measured and expected peak positions in reciprocal space. Figure 5 shows the input panel for the calibration operator, and illustrates which instrument parameters can be calibrated.

Operation SCDcalib

Peaks file

/home/dennis/TEST_ISAW_

Browse

Run file

/home/dennis/TEST_ISAW_

Browse

lattice 'a'

4.9138

lattice 'b'

4.9138

lattice 'c'

5.4051

lattice 'alpha'

90.0

lattice 'beta'

90.0

lattice 'gamma'

120.0

max steps

500

tolerance exponent

-12

☒ Refine L1

☒ Refine t0

☐ Refine 'A'(tof=At+t0)

☐ Refine sample shift

☒ Refine det width

☒ Refine det height

☒ Refine det x_offset

☒ Refine det y_offset

☐ Refine det distance

☐ Refine det rotation

☐ LOAD INITIAL VALUES FROM FILE

ALL PARAMETERS FILE

SCDcalib.results

Browse

Result

Apply

Exit

Help

Figure 5

Reciprocal Lattice Viewer

The Wizards written for single crystal diffraction provide an easy to use interface, and work well with a set of strong peaks from a single crystal. Unfortunately, this is not always the case. If, for example, the crystal is "twinned", the basic auto indexing routine is likely to fail. In order to visualize this and allow the user to choose and index one crystallite at a time from a twinned crystal, a 3D reciprocal lattice viewer was designed to let the user interactively select families of planes to use for indexing the peaks within the reciprocal lattice.

The essential capabilities of the reciprocal lattice viewer include:

- Display voxels in reciprocal space, corresponding exactly to time-of-flight histogram bins from the area detectors.
- Allow user selection of families of planes of peaks in reciprocal space.
- Use Fourier Transforms of projections of peaks in various directions to assist the user in selecting planes. (This is similar to the Rossmann indexing algorithm[2][3].)
- Allow user to restrict data to a family of planes in reciprocal space.
- Calculate an orientation matrix, when three independent families of planes have been chosen.

The raw time-of-flight data from the SCD at IPNS consists of 20,000 time-of-flight histograms, one histogram for each pixel on the area detectors. Each time-of-flight histogram bin corresponds to a different neutron wavelength/energy which can be determined from the time-of-flight of the neutron over the known flight path. Thus each time-of-flight histogram bin corresponds to an element of volume in reciprocal space. For each time-of-flight histogram bin for which the counts exceed a specified threshold, the reciprocal lattice viewer maps eight points corresponding to the four corners of the pixel in real space and the beginning and ending times-of-flight, to reciprocal space. These eight points define a distorted "box" in reciprocal space. Each such box is drawn in a color corresponding to the number of counts in that histogram bin.

If the crystal is, in fact, single, the peaks will fall on families of planes in reciprocal space. In order to find basis vectors for a primitive unit cell in real space, it is sufficient to find three independent families of planes in reciprocal space with the three largest inter-plane spacings. Typically, at least the family of planes with the largest inter-plane spacing in reciprocal space is relatively easy to identify.

The reciprocal space viewer allows the user to interactively choose a family of planes by selecting three peaks. The peaks are selected by clicking on them and then pressing the "Select", "Select +" or "Select *" buttons. "Select" selects a new origin. "Select +" or "Select *" specify two additional points, which form vectors drawn from the origin to the newly selected points. The "Q" value for the origin is displayed, as are the "Q" components and lengths of the two vectors that are formed. See Figure 6.

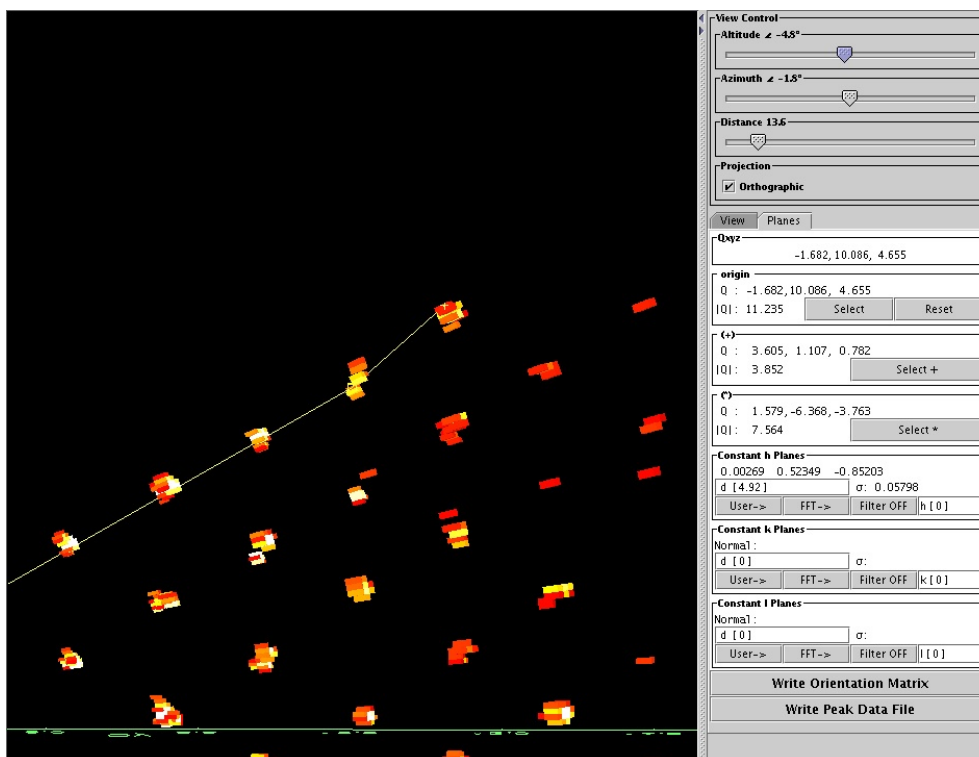


Figure 6

The positions of the three selected peaks determine a plane normal. The user can choose to interpret this family of planes as planes of constant h , k or l values by pressing the "User->" button in the appropriate plane control. Suppose that the user chooses to interpret the specified family of planes as "Constant h Planes". After pressing the "User->" button, the cross product of the vectors is calculated, normalized and displayed in the "Constant h Planes" control. Given this normal, the software projects all peaks onto a line in the direction of the normal. If the peaks lie on a family of planes with that normal, the projections will form a regular pattern. The projection is Fourier transformed to identify the fundamental frequency in the projection of the peaks. Based on this fundamental frequency and the normal direction, integer " h " values are assigned to each peak, and a refined normal direction is determined by finding the least squares solution to the over determined system of equations:

$$\vec{n} \cdot \vec{q}_i = h_i, i = 0, 1, 2, \dots, N$$

where \vec{q}_i is one of the N voxels with counts above the required threshold in reciprocal space and h_i is the assigned " h " value. The fundamental frequency in this pattern is also mapped to real space and the corresponding d -spacing is displayed in the "Constant h Planes" control. The user can also enter a value for " d " in this control, if the crystal lattice parameters are known and the calculated value for " d " is not sufficiently accurate.

Having selected a family of planes, the user may choose to discard all peaks that are sufficiently far from all planes in the family of planes. A default tolerance of 10% of the plane spacing is used. If the data contains spurious peaks from any source, approximately 80% of the spurious peaks should be discarded, and virtually none of the desired peaks will be discarded. This "filter" operation can be turned on or off using the "Filter On/Off" button in the plane control.

In this way, the user can manually choose three families of planes in reciprocal space, filtering to any or all of the three families of planes, as needed to omit spurious peaks.

To assist in the process of choosing families of planes, the software will determine a list of approximately 600 unit vectors whose endpoints are spread uniformly over a unit hemisphere. For each of these possible plane normals, the peaks are projected onto the normal direction, Fourier transformed and refined as described above. From the resulting Fourier transforms a smaller set of at least 10 Fourier transforms are selected based on various heuristics, such as the residual error from the least squares refinement. The selected set is ordered by increasing "d-spacing", and displayed as rows in an image. When the user points at a row in this image, the d-spacing is displayed on the border of the graph below the image. The corresponding family of planes is also indicated in the reciprocal space viewer, by drawing a sequence of boxes along the normal direction, with the space between successive boxes equal to the spacing between the planes. See Figure 7.

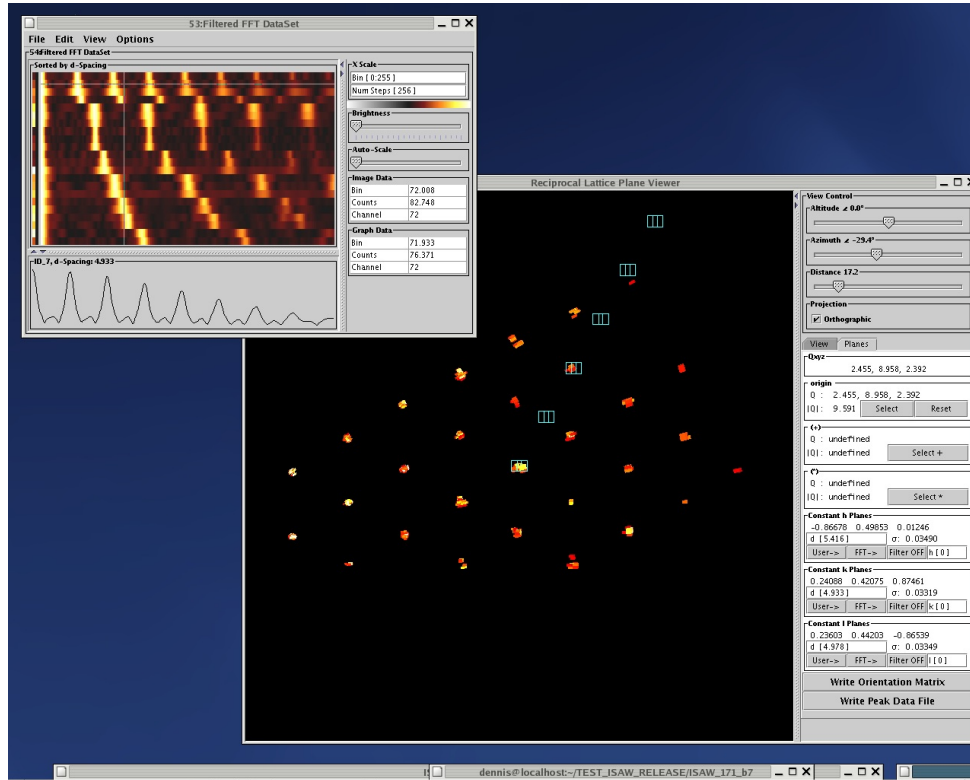


Figure 7

After finding a set of possible families of planes in this manner, it is fairly easy for the user to step down the rows of the image, noting the d-spacing and examining the suggested family of planes in reciprocal space. The user can choose to use such a family of planes as constant h, k or l planes by pressing the "FFT->" button. As before, the calculated value of "d" can be replaced by a more accurate value, if the lattice parameters are already known.

When the user has chosen constant h, constant k and constant l planes, integer (h,k,l) values can be assigned to each voxel, \vec{q}_i , with counts above the currently specified threshold. Using these assigned (h,k,l) values, the orientation matrix can be calculated. The orientation matrix is calculated as the the matrix M that most nearly maps each of the triples

$$\vec{v}_i = (h_i, k_i, l_i)$$

to the corresponding voxels

$$\vec{q}_i = (qx_i, qy_i, qz_i).$$

That is, a least squares method is applied to find coefficients for the orientation matrix, M , so that

$$M \vec{v}_i = \vec{q}_i,$$

as nearly as possible.

At several stages in the SCD software, it was necessary to solve a linear least squares problem. The solution to such least squares problems are often described in terms of the "normal equations". Unfortunately, the normal equations can be ill-conditioned and so may be difficult to solve accurately. A more stable solution can be obtained by using Householder[5] transformations (elementary reflectors) to reduce the matrix to upper triangular form[6]. Since the Householder transforms preserve distances, errors are not magnified by the solution process. This method of solving the least squares problem is used throughout ISAW.

Conclusions

ISAW provides a large set of classes for neutron scattering data visualization and analysis. The support for single crystal diffractometers includes a powerful and novel interactive viewer for reciprocal space that allows the user to interactively select families of planes in reciprocal space. This process has been successfully used to deal with crystals that are twinned. The software can be downloaded from the IPNS website, <http://www.pns.anl.gov/computing/ISAW/>

Acknowledgment

The authors would like to acknowledge the contributions of their colleagues at IPNS and their students. In particular, Dr. Peter Peterson, ORNL, made significant contributions to the underlying data analysis algorithms for single crystal diffraction while he was a post-doc at IPNS. Also, Chris Bouzek did much of the implementation of the SCD "Wizards" while he was a student at the University of Wisconsin-Stout.

The authors also gratefully acknowledge the support for the development of the single crystal software in ISAW by the National Science Foundation, under grant number DMR-0218882. Work at Argonne was funded by the U.S. DOE-MS under contract number W-31-109-ENG-38.

References:

- [1] GNU GPL, <http://www.gnu.org/copyleft/gpl.html>
- [2] Powell, H.(1999). Acta Cryst., **D55**, 1690-1695.
- [3] Steller, I., Bolotovskiy, R. & Rossman, M.G.(1997). J. Appl.Cryst. **30**, 1036-1040.
- [4] Bevington, P., Robinson, D.(1992). Data Reduction and Error Analysis for the Physical Sciences, (WCB McGraw-Hill, Boston)
- [5] Householder, A.(1975). The Theory of Matrices in Numerical Analysis, (Dover, New York)
- [6] http://sep.stanford.edu/sep/prof/fgdp/c6/paper_html/node5.html

Graphical and interpretation tools for difficult incommensurate and composite structures in JANA2000

Václav Petříček and Michal Dušek,

Institute of Physics of ASCR, Na Slovance 2, 182 21 Praha 8, Czech Republic - Email : petricek@fzu.cz and <mailto:dusek@fzu.cz> ; WWW: <http://www-xray.fzu.cz/jana/>

Introduction

The increase in solved modulated structures and the latest development of the methods for their solution is closely connected with advances in the instrumentation. Modern diffractometers give more or less complete diffraction pattern of the crystal. This fact and their very good sensitivity minimizes the chance that satellites would be overlooked or too weak for measurement. Moreover data collection programs for most of commonly used diffractometers allow integration of these additional reflections.

Existence of satellite reflections is directly connected with fact the classical 3d translation symmetry in the modulated crystal is lost. The atoms from cell to cell change their basic structural parameters such as occupancies, positions and ADP (atomic displacement parameters). However, these changes are not random; they can be described by periodic modulation functions:

$$u(\mathbf{q}_1 \cdot \mathbf{r}, \mathbf{q}_2 \cdot \mathbf{r}, \dots, \mathbf{q}_d \cdot \mathbf{r}) = u(\mathbf{q}_1 \cdot \mathbf{r} + n_1, \mathbf{q}_2 \cdot \mathbf{r} + n_2, \dots, \mathbf{q}_d \cdot \mathbf{r} + n_d)$$

The modulation function must be reasonably smooth function with maximally a few discontinuities. Then the structure can be described using basic functions with periodic perturbations.

The fact that the diffraction pattern is still made from clearly distinguished diffraction spots was used by deWolff, Janssen, and Janner [1] to develop the so called "superspace approach". Additional vectors perpendicular to regular three dimensional space were introduced to recover the translation symmetry in the more dimensional elementary cell. This general approach made it possible to generalize structure determination techniques for their application in (3+d) dimensional space. The superspace concept provides us also with a visualization method to demonstrate real modulations in the crystal.

The crystal structure analysis of modulated and composite crystals is becoming more or less standard. The modulation parameters for various building units of the structure can take many different functional forms. For example the strong step-like modulation in one structural unit may induce a smoother modulation in the rest of the structure. A Fourier synthesis and animation techniques play very important role during structure determination and refinement. The purpose of this contribution is to present here these methods.

Fourier (3+d) dimension techniques

The generalized density of diffracting objects (electrons for X-rays, nuclei for neutrons) in the modulated structures has (3+d) dimensional periodicity. This means that it can be expressed as a 3+d dimensional Fourier series:

$$\bar{\rho}(\mathbf{R}) = \sum_{\mathbf{H}} \bar{F}(\mathbf{H}) \exp(-2\pi i \mathbf{R} \cdot \mathbf{H}),$$

where \mathbf{H} and \mathbf{R} are respectively diffraction and positional (3+d) vector and $\bar{F}(\mathbf{H})$ are generalized structure factors related by the standard way to the integrated intensities. The maps can be calculated either when phases are already known at least in some approximation or when the above equation is used for $\bar{F}^2(\mathbf{H})$. For the latter case the Patterson maps are obtained. They can be used to find a starting modulation model

in cases when the structure contains some dominating heavy atoms. In the Fig 1 the modulation of one interatomic vector between two symmetrically related positions of heavy atom is presented. The periodic expansion and contraction of density is induced by a mutual modulation of two atomic positions in the selected direction. In the case when the modulation is negligible the density section would show just a uniformly distributed density. The amplitude can be estimated from the difference between the most expanded and the most contracted area. The x_4 position corresponding to the most contracted density can help to find the phase of the modulation function. For more details see the original work [2].

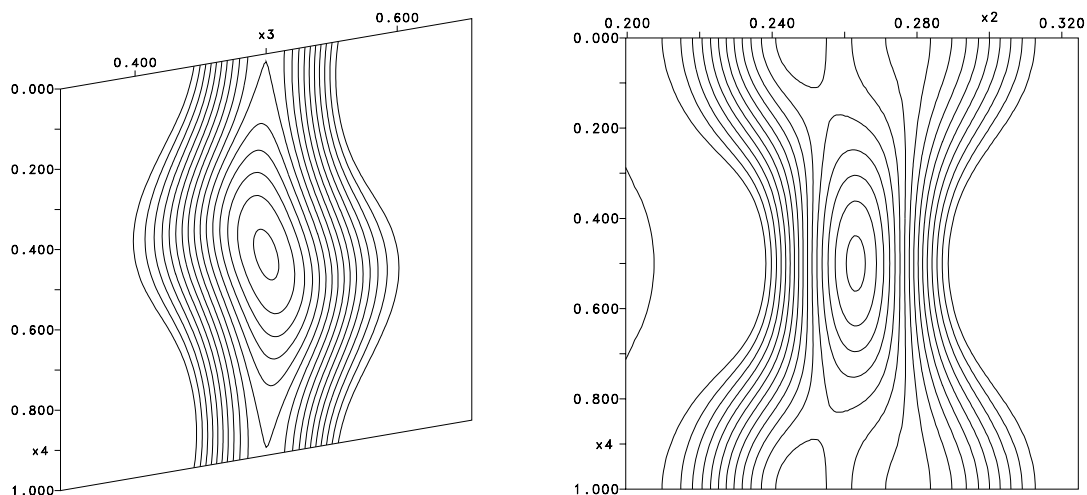


Fig. 1: $x_2 - x_4$ and $x_3 - x_4$ sections of the Patterson function of heavily modulated atom

The Fourier maps based on phases derived from the modulation of the heavy atom can be used to find a modulation curve of light atoms (see Fig 2).

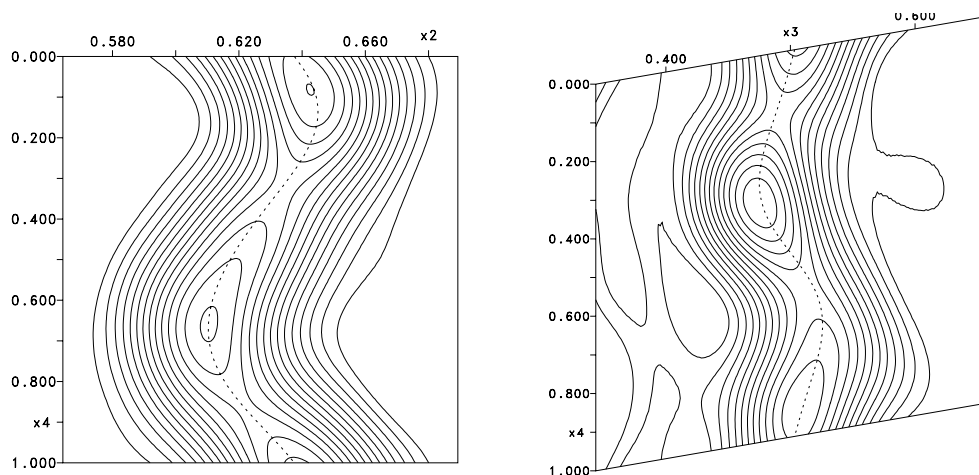


Fig 2: The $x_2 - x_4$ and $x_3 - x_4$ sections of Fourier maps in the vicinity of a light atom

Fourier maps can also reveal a special character of the modulation in cases when the positional or occupational modulation has a discontinuous character. As an example we use the modulated structure of $\text{Cd}(\text{NH}_3)_3\text{Ni}(\text{CN})_4$ [3]. In the **fig.3** it is shown how the modulation of Cd atoms in this structure look like:

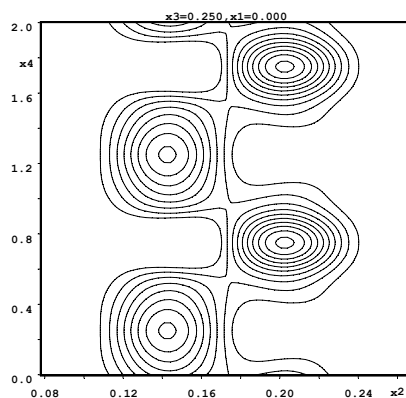


Fig 3: The occupational modulation of Cd1 and Cd2. The two basic positions are clearly separated into two crenel-like intervals.

The map shows clearly that the cadmium atom occupies two different positions. More detailed analysis shows that one position is octahedrally coordinated while the other is tetrahedrally coordinated. The number of harmonic functions to model satisfactory such a modulation would be very high. The crenel-like approach [2] can considerably reduce number of parameters.

The Fourier sections of type $x_n - x_4$ presented hitherto show clearly changes of a certain coordinate of a selected atom as a function of the modulation coordinates. This is very important for finding the best modulation model for one particular atom but, on the other hand, these maps cannot usually give a direct idea about the structure in three dimensions. The best way to present mutual interactions in modulated crystals is to draw three-dimensional maps as a function of the internal coordinate t . In the next figure we demonstrate usefulness of properly selected section and/or projections in the structure $\text{Cd}(\text{NH}_3)_3\text{Ni}(\text{CN})_4$ [3]. The two-dimensional section running through the $\text{Ni}(\text{CN})_4$ group shows how the step-like modulation of cadmium affects the other atoms:

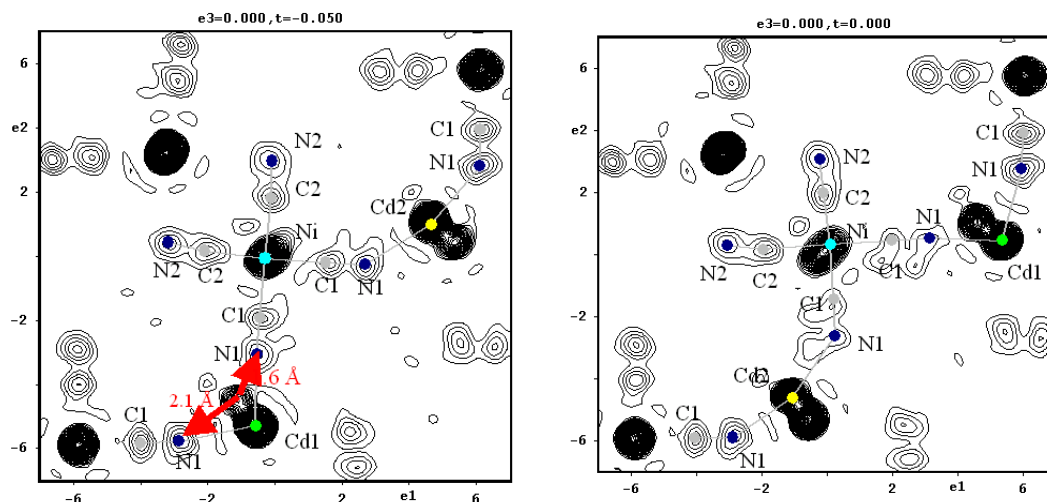


Fig 4: Sections through the four-dimensional Fourier map showing the coordination of cadmium by two symmetry-related cyano groups C1-N1. The section runs through the plane of $\text{Ni}(\text{CN})_4$ in the basic structure; $e1, e2, e3$ are the Cartesian axis.

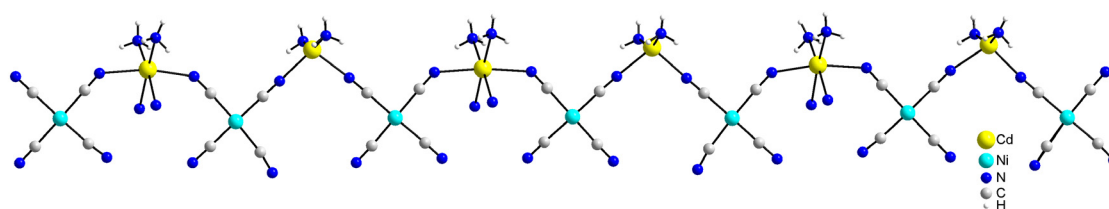
The first section is plotted for the internal coordinate $t=-0.05$, which is near of the refined end point of the crenel function definition interval, i.e. it shows the situation just before the "jump". The atom Cd1 (green) still keeps the tetrahedral coordination but the new octahedral position (Cd2) is already arising. Distances between the new (arising) Cd2 position and two neighboring atoms N1 are respectively 1.6 Å and 2.1 Å.

The too short first distance (1.6 Å) forces an abrupt change in the coordination that is visible in the second section calculated for $t=0$. Here the Cd atoms are already localized in the new octahedral position (Cd2) and the cyano group C1-N1 on the right is shifted to achieve more realistic distance to Cd2.

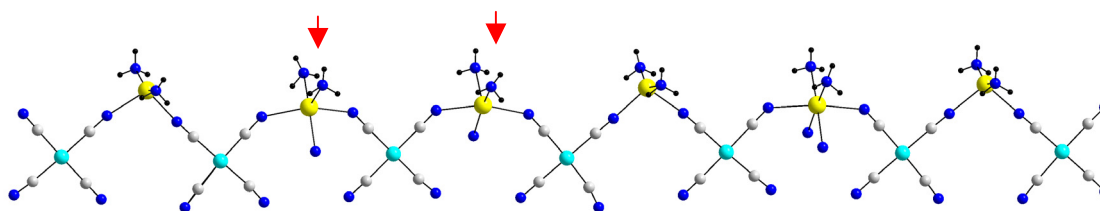
Graphic interpretation of structural results

Till now there has been no common program which can directly use superspace group information to draw modulated structures. Therefore modulated positions of atoms must be pre-calculated in some artificial large cell beforehand. Then a standard drawing program such as ATOMS or DIAMOND can be used to produce figures. As the modulated structure is not periodic we have to suppress in the plotting software generating of atoms outside of the pre-calculated area in the modulation direction(s). In Fig. 5 we show representation of modulated structure plotted from pre-calculated coordinates by the program Diamond [].

Fig 5: *The modulation along the chain $[-\text{Cd}(\text{NH}_3)_n\text{-NC-Ni}(\text{CN})_2\text{-CN-}]_\infty$*



(a) *The most common situation: cadmium has alternatively octahedral and tetrahedral coordination.*



(b) *An intermediate states in which cadmium exhibits penta-coordination are indicated by the red arrows*

Graphical and interpretation tools in Jana2000

Program Jana2000 is based on the generalized crystallography following from the superspace approach. As its main features, we should mention possibility to use single crystal or powder diffraction data originated from the X-ray, synchrotron or neutron experiment; refinement of modulation of occupancy, position, harmonic and anharmonic ADP; refinement of anomalous dispersion coefficients; multipole refinement; automatic setting of symmetry restrictions for refined parameters, user constraints and restraints, powerful rigid body access; calculation and visualization of 3+d dimensional electron density maps. The number of modulation vectors is limited to three according to current experience. The access to three or more dimensional structures is unified, i.e. the user sees the same tools regardless of the dimension. The underlying idea of the program is to offer a simple way for solving simple tasks but keep all possibilities open for complicated structures.

Data Input. The single crystal input data can be provided either as a rough diffractometer file or as a file already processed by some data reduction software. Data from various sources can be combined provided that the wavelength of the partial data is the same. The reading of data can be followed by automatic transformation that changes number of indices according to the target dimension. For instance, three real indices used for measurement can be automatically transformed to integer indices of modulated structure. Any user defined transformation of indices is possible as well as a transformation resulting from supercell

definition. During import of twin domains the program can automatically reveal merohedry by tentative transformation of indices using the user defined twin matrices, see Fig. 6. For powder data the program supports several basic input forms like GSAS, FullProf and many others. Jana2000 cannot index powder data. The *symmetry information* can be entered by a tool shown in Fig. 7. using a symbol of the (super)space group or list of symmetry operators. Non-standard settings and centring are widely accepted. Several tools for transformation of the structure model are available, especially cell transformation, group-to-subgroup transformation and commensurate structure-to-supercell transformation. In all cases the complete structure model is transformed including coordinates, cell parameters, indices and symmetry. The tool for group-to-subgroup transformation is presented in the figure 8.

Fig. 6: Example of data input to Jana2000. A three-fold twin of CsLiSO₄ was indexed in a supercell. The transformation matrix transforms the supercell into the finally used elementary cell. The overlap option causes that the reflections are sorted automatically into the first, second or third twin domain using the twinning matrices.

Fig. 7: Tool for entering symmetry information into Jana2000. The symmetry can be entered by symbol or symmetry operators. "Complete the set" completes set of operators to form a group and derives the (super)space group symbol from the operators. The origin can be arbitrarily shifted. In addition to standard centering symbols a non-standard centering *X* can be used with user defined centering vectors.

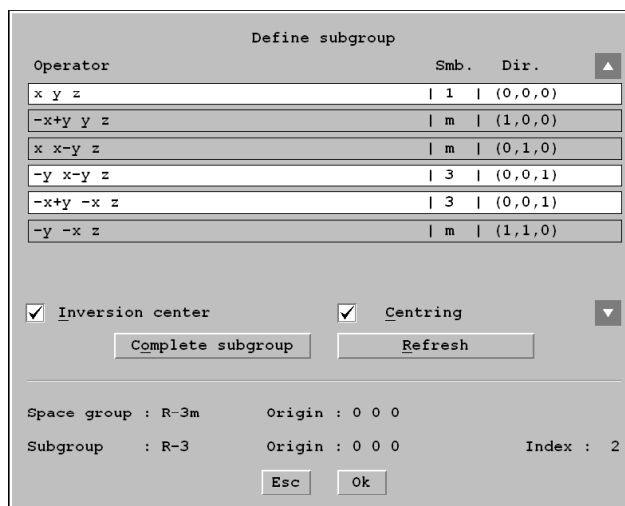


Fig. 8: Tool for group-subgroup transformation. The white lines are the symmetry operations selected from the set of existing operators. In this case the inversion center and the centering are retained. If they are removed the list of existing operators available for selection is expanded. "Complete subgroup" derives the (super)space group symbol from the selected operators. Then the structure model is transformed and expanded into the subgroup. The removed symmetry operations can be used as twinning operations.

Structure solution and refinement.

Jana2000 cannot automatically solve the phase problem. Instead, it prepares an input for external direct methods programs. The refinement program can make Le Bail refinement of profile parameters, Rietveld refinement based on powder data or classical structure refinement based on single crystal data. For powder data it also refines modulation vectors. Refinement of profile parameters includes Gaussian, Lorentzian or Pseudo-Voigt profiles and the following corrections: anisotropic particle and strain broadening, profile asymmetry, preferred orientation, absorption and background. Structure refinement offers the same possibilities for both single crystal and powder data. For all structures refinement is possible of position, occupation, twin fractions, extinction, isotropic, anisotropic and anharmonic ADP. For 3d structures the multipole refinement is available, too. Occupation, position and ADP parameters can be modulated by harmonic modulation waves or by discontinuous modulation functions for occupation (crenel function) and for position (sawtooth function).

The refinement program sets automatically symmetry restrictions of refinable parameters. Parts of refined structure can be refined within the rigid body approximation in one or more positions in the elementary cell. The rigid body approach as used in Jana2000 allows refinement of occupation, position and TLS modulations for each rigid body position and combination of rigid body and free atomic contributions for selected parts of rigid body. Structure parameters can be fixed or made dependent using their unique identifiers and tools for setting refinement options. The simplest option is fixing a parameter or group of parameters to zero or to their actual value. The user can also define linear dependencies between parameters or require the same modulations, temperature parameters or sum of occupancies for two or more atoms. Selected bond lengths and angles can be restrained to a desired value within a user defined limit based on their standard uncertainty. Set of geometrical constraints is available for fixing a geometry of some atomic group, keeping structural fragments planar and forcing positions of some atoms to complete a defined geometric shape. The latter is especially useful for positions of hydrogens, which should complete a tetrahedral or trigonal coordination given by remaining non-hydrogen atoms, see Fig. 9. The bond length and angle restraints as well as the geometrical constraints work for both standard and modulated structures.

Jana2000 enables calculation and visualization of Fourier maps. It can also pre-calculate atomic positions for visualization of modulated structures by an external plotting program. Both these features were discussed in section 2 and 3. Moreover, the program contains a tool for plotting various structure parameters as a function of t coordinate, for instance position, occupancy, ADP parameters, distances and angles. A growing importance has interpretation of bond valences that is good indication of stability of coordination of an atom in modulated structure. An example is given in Fig. 10.

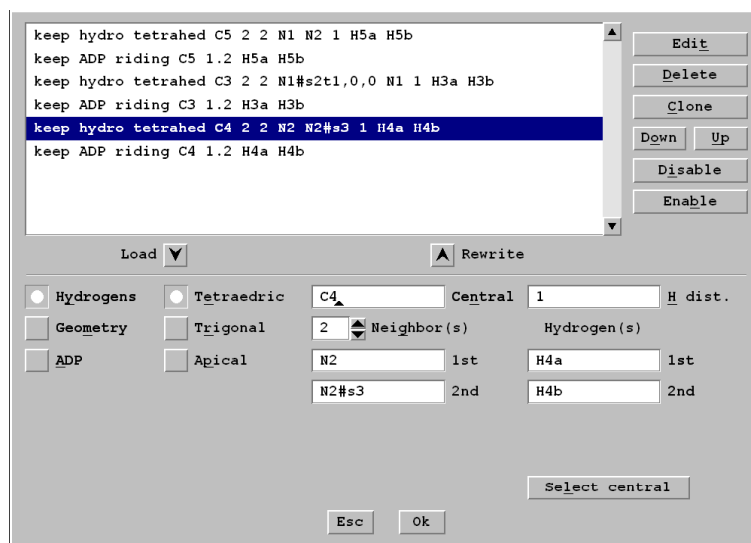


Fig. 9: Tool for constraining refinement of hydrogens. The highlighted option keeps two hydrogen atoms, H4a and H4b, to complete tetrahedric coordination around the central atom C4. It should be noted that this constraint works also for modulated structures.

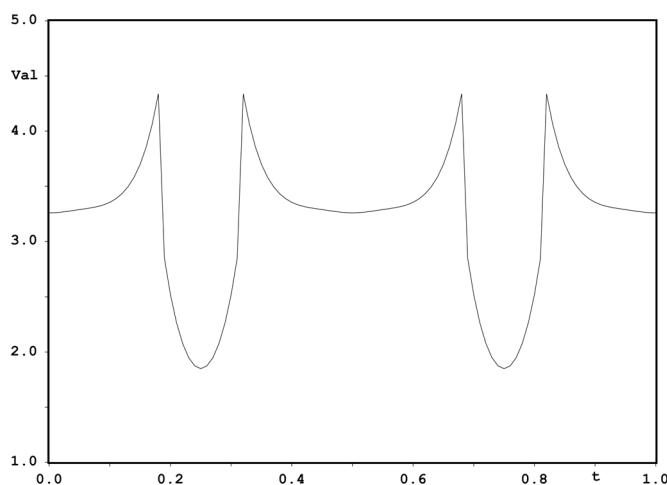


Fig. 10: Bond valence sum calculated for the environment of Co in Sr_{1.274}CoO₃ [5] as a function of t . Large abrupt changes of the bond valence sum correspond with the discontinuous changes of cobalt coordination found in this structure.

References:

- [1] Wolff, P.M. de, Janssen, T. & Janner, A. (1981). *Acta Cryst.*, **A37**, 625-636.
- [2] Petříček, V., van der Lee, A. & Evain, M. (1995). *Acta Cryst.* **A51**, 529-535.
- [3] Petříček, V., Dušek, M. & Černák, J. (2005). *Acta Cryst.* B61, to be published
- [4] Petříček, V. & Dušek, M. (2000). *JANA2000. Programs for Modulated and Composite Crystals*. Institute of Physics, Praha, Czech Republic.
- [5] Evain, M. ; Boucher F. ; Gourdon O. ; Petříček V. ; Dušek M. ; Bezdička, P. (1998). *Chem.Mater.*, (1998), **10**, 3068-3076.

Calculating the Pair Distribution Function from a Structural Model

Thomas Proffen

Lujan Center, Los Alamos National Laboratory, MS H805, Los Alamos, NM 87544, USA Email: tproffen@lanl.gov - WWW: <http://lansce.lanl.gov/lujan/ERIER2/NPDF/> ; LA-UR 05-0406

Introduction

The determination of crystal structures is an important part of chemistry, physics and of course crystallography. Conventional structure determination is based on the analysis of the intensities and positions of Bragg reflections which only allows one to determine the long range **average** structure of the crystal. Only one-body information such as atomic positions, site occupancies and temperature factors can be extracted. Determination of the average structure based on powder diffraction data is now routinely done using the Rietveld method. It should be kept in mind, however, that the analysis of Bragg scattering assumes a perfect long range periodicity of the crystal. However, many materials are quite disordered and even more importantly, the key to the deeper understanding of their properties is the study of deviations from the average structure or the study of the **local** atomic arrangements. Deviations from the average structure result in the occurrence of diffuse scattering which contains information about two-body interactions. A convenient way to reveal the local, medium and long range structure of a material is the analysis of the Pair Distribution Function (PDF). The PDF is obtained via Fourier Transform from properly normalized diffraction intensities. This method has long been applied to the study of short range order in liquids and glasses but has recently been extended to crystalline materials. A summary of this technique and its applications is given in the review by Proffen et al. [1] as well as the recent book by Egami and Billinge [2].

The key to the successful analysis of the PDF is often the ability to refine a structural model to the experimental data. Two programs to calculate and refine PDFs from a structural model are the real space refinement program PDFFIT [3] and the general defect structure simulation program DISCUS [4,5]. Both programs are available from the author or on the WWW at <http://www.totalscattering.org/programs/discus/>. This paper gives a short summary of the details, how these programs calculate the PDF given an atomic structure.

Calculating the PDF

The PDF can simply be understood as a bond-length distribution of the material under investigation, in other words the PDF is related to the probability of finding an atom A at a distance r from an atom B as illustrated in Figure 1. Unfortunately there are many different definitions used in the crystalline, glass and liquids community as well as different program packages. A summary of definitions and conversions is given in a paper by Keen [6].

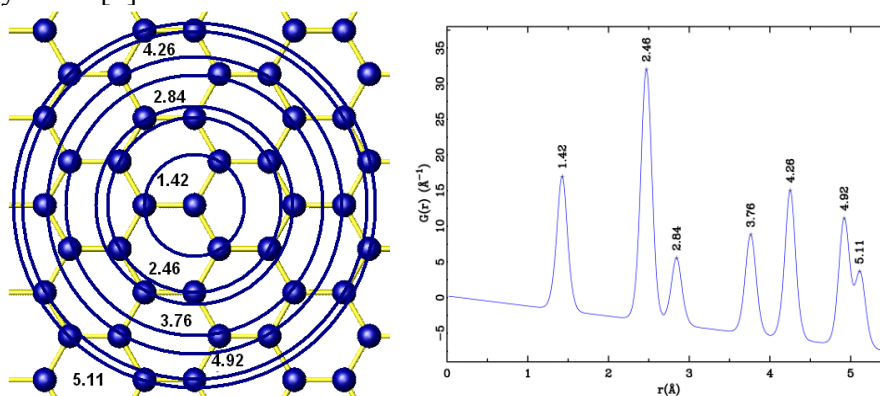


Figure 1: Schematic diagram of the PDF of graphite. The circles mark the near-neighbor shells. The corresponding distances can easily be seen in the PDF. Distances are given in Å.

In this paper we will use $G(r)$ which is implemented in the programs DISCUS and PDFFIT and is in some way the PDF function of choice for disordered crystalline materials [6]. The function is calculated from a structural model using the relation

$$G(r) = \frac{1}{r} \sum_i \sum_j \left[\frac{b_i b_j}{\langle b \rangle^2} \delta(r - r_{ij}) \right] - 4\pi r \rho_0. \quad (1)$$

In this equation, the sums loop over all atoms i and j in the structure which are separated by a distance r_{ij} . In addition each contribution is weighted by the atoms scattering power b_i . Finally $\langle b \rangle$ is the average scattering power of the sample and ρ_0 is its number density. In practice, the first sum loops over all atoms i in the crystal, but the second sum is limited to neighboring unit cells with a distance not larger than the probing distance r_{ij} . This reduces the computing time significantly, but it requires the program to maintain the information relating a particular atom to its unit cell. In DISCUS and PDFFIT this is realized via a specific order of the atoms within the computer memory. As the program calculates the contributions of each relevant atom-atom pair, the corresponding weight is ‘histogrammed’ on a specified grid in r .

Introducing thermal motion

In reality atoms move due to thermal motion, resulting in a broadening of the PDF peaks related to the atomic displacement parameters (adp) of the respective atoms. Calculating the PDF following equation (1) will give one sharp contribution for each atom-atom pair sampled. In cases where the model crystal is of sufficient size the atoms can be randomly displaced according to their adp’s. As a result

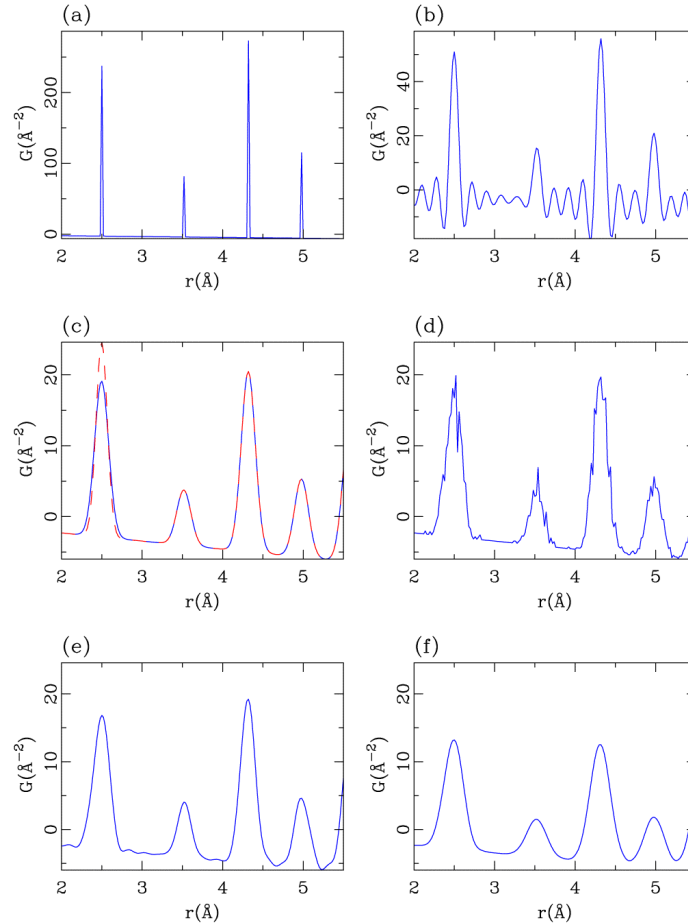


Figure 2: Modeling of thermal motion: (a) no thermal motion; (b) no thermal motion, $Q_{\max}=35\text{\AA}^{-1}$; (c) convolution with Gaussian (blue solid line) and correlated motion (dotted red line); (d) ensemble average 3x3x3 unit cells; (e) Gaussian, $Q_{\max}=35\text{\AA}^{-1}$ and (f) ensemble average and convolution.

the calculated PDF peaks will show a reasonable width. On the other hand in many cases the disorder in the model can be described sufficiently using a much smaller model. In this case, an alternative way to introduce thermal broadening is needed. The simplest way is to convolute the expression in Equation 1 with a Gaussian of appropriate width. This is illustrated in Figure 2: (a) shows the PDF of a perfect crystal with no thermal displacements and as a result, sharp peaks are observed for the different atom-atom distances. In Figure 2c we see the PDF of the same model crystal, only this time thermal motion was modeled using Gaussians. The alternative of a larger model crystal (here 3x3x3 unit cells) with individually displacing atoms is seen in Figure 2d. Because of the still limited size of the model crystal, the PDF peaks appear noisy. The other parts of Figure 2 deal with termination effects and are discussed in the next section.

Thorpe and co-workers [7] have shown that the pure Gaussian is insufficient due to anisotropic averaging and the function actually used in PDFFIT and DISCUS to model thermal motion is

$$T_{ij}(r) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp\left[-\frac{(r-r_{ij})^2}{2\sigma_{ij}^2}\right] \left[1 + \left(\frac{r-r_{ij}}{r_{ij}}\right)\right]. \quad (2)$$

Here σ_{ij} is the peak width calculated from the respective adp's of atoms i and j . As you can see the last term is the correction to the Gaussian PDF peak shape mentioned above.

Termination effects

As mentioned above, the experimental PDF is obtained via a Fourier Transform and as a result, the observed PDF contains truncation contributions. Although there are many approaches using dampening functions before applying the Fourier Transform to minimize these contributions, the effect of a cutoff at a given value Q_{max} can be modeled analytically. All that is required, is a convolution of the model PDF with the Fourier Transform of the step function terminating the diffraction data. This function is

$$S(r) = \frac{\sin(Q_{max}r)}{r}. \quad (3)$$

In practice it turns out that for large Q_{max} , the termination effects are very small compared to the contribution due to noise in the diffraction data. Values of $Q_{max} > 40\text{\AA}^{-1}$ can now routinely be achieved at instruments located at synchrotron or spallation neutron sources.

The r -dependence of the PDF peak width

The final piece needed to calculate a PDF from a structural model are corrections due to instrument resolution as well as correlated motion. Let us discuss correlated motion first: Near neighbor atoms have the tendency to move in phase causing near neighbor PDF peaks to sharpen compared to far neighbor peaks. The size of the effect depends on the bonding in the crystal and e.g. for a covalent bonded semiconductor alloy this sharpening is much more pronounced than for example for a metal as discussed in a paper by Jeong et al. [8]. The second effect to be discussed is the instrument resolution. It causes an increase of the PDF peak width as function of distance r as well as an exponential dampening of the PDF peaks as function of r . An example is shown in Figure 2c when comparing the PDF shown in blue with no correction for correlation motion with the PDF shown in red showing a sharper nearest neighbor PDF peak due to correlated motion. The r dependence of the PDF peak width as implemented in the programs DISCUS and PDFFIT is given by

$$\sigma_{ij} = \sqrt{\sigma'_{ij} - \frac{\delta}{r_{ij}^2} - \frac{\gamma}{r_{ij}} + \alpha^2 r_{ij}^2} . \quad (4)$$

Here σ'_{ij} is the peak width calculated from the adp's of the model crystal and r_{ij} is the atom-atom distance. Equation (4) reveals the two correction terms δ and γ to account for correlated motion. The different r dependence is related to the vibrational model suitable for the studied system and details can be found in [8]. The correction term α accounts for peak broadening at large r due to instrument resolution. In practice, however, this effect is only observed for distances above 20Å on medium- and high resolution instruments.

For a more detailed discussion, please refer to the PDFFIT users guide [9] and it desired the source code of PDFFIT and DISCUS which is part of the programs UNIX distribution.

References

- [1] Th. Proffen, S.J.L. Billinge, T. Egami and D. Louca, **Structural analysis of complex materials using the atomic pair distribution function - a practical guide**, *Z. Krist.* **218**, 132-143 (2003).
- [2] T. Egami and S.J.L. Billinge, **Underneath the Bragg-Peaks: Structural Analysis of Complex Materials** , Elsevier Science B.V., Amsterdam, 2003.
- [3] Th. Proffen and S.J.L. Billinge, **PDFFIT a Program for Full Profile Structural Refinement of the Atomic Pair Distribution Function**, *J. Appl. Cryst.* **32**, 572-575 (1999).
- [4] Th. Proffen and R.B. Neder, **DISCUS a Program for Diffuse Scattering and Defect Structure Simulations** , *J. Appl. Cryst.* **30**, 171-175 (1997).
- [5] Th. Proffen and R.B. Neder, **DISCUS a Program for Diffuse Scattering and Defect Structure Simulations - Update** , *J. Appl. Cryst.* **32**, 838-839 (1999).
- [6] D.A. Keen, **A comparison of various commonly used correlation functions for describing total scattering**, *J. Appl. Cryst.* **34**, 172-177 (2001).
- [7] M.F. Thorpe; V.A. Levashov; M. Lei; S.J.L. Billinge. **Notes on the Analysis of Data for Pair Distribution Functions**. In S. J. L. Billinge; M. F. Thorpe, editors, *From Semiconductors to Proteins* page 105 New York 2002. Plenum.
- [8] I.-K. Jeong, Th. Proffen, F. Mohiuddin-Jacobs and S.J.L. Billinge, **Measuring Correlated Atomic Motion using X-Ray Diffraction**, *J. Phys. Chem. A* **103**, 921-924 (1999).
- [9] I.-K. Jeong, R.H. Heffner, M.J. Graf and S.J.L. Billinge, **Lattice Dynamics and Correlated Atomic Motion from the Atomic Pair Distribution Function**, *Phys. Rev. B* **67**, 104301 (2003).
- [10] Th. Proffen and S.J.L. Billinge, **PDFFIT Users Guide, Manual Version 1.3** (2003).

Refinement in Crystals

Richard Cooper and David Watkin,

Chemical Crystallography, Department of Chemistry, University of Oxford, Chemistry Research Laboratory, Mansfield Road, Oxford, OX1 3TA, UK - Email : david.watkin@chem.ox.ac.uk ; WWW: <http://www.xtl.ox.ac.uk/> and <http://www.chem.ox.ac.uk/researchguide/djwatkin.html>

In the last newsletter, there were brief articles on refinement in both SHELXL97 and *SIR2004*. We have been invited to describe how CRYSTALS fits in with them. The authors of these programs and of CRYSTALS have all known each other for many years and enjoy and benefit from the amicable competition between their programs. To complete the stories started by Sandy Blake and the *SirTeam* last time, here follows a brief description of where CRYSTALS fits in with these other excellent programs.

During the 1979's the Oxford Chemical Crystallography Laboratory had a strong interest all aspects on crystallographic computing, including Direct Methods, with a good Symbolic Addition program being written by John Hodder, and Tom Blundell's tangent refinement program. At that time Davide Viterbo and Edwardo Castellano also made their contributions as post docs. Because of Keith Prout and John Rollett's involvement in twinning, the main thrust of programming moved into refinement and analysis, which remains our strongest interest. We now have no skills in programming Direct Methods, but fortunately we are permitted to distribute re-compiled versions of SHELXS 84 by George Sheldrick and SIR92 by Carmelo Giacovazzo along side CRYSTALS. Interfaces exist in CRYSTALS to enable it to communicate freely with more modern Direct Methods programs which users can obtain directly from the authors. Structure refinement and analysis remain our principal domains.

CRYSTALS – the users view

It is now common for serious computing environments to have dual user interfaces – a GUI for the more routine tasks, and a Command Line Interpreter (CLI) which enables the user to get into close communication with the full richness of the system. A similar duality exists for CRYSTALS. The CLI, which was the only mechanism for communicating with the program in the days of mainframe computing, has been retained and extended, and offers the users access to an amazing wealth of operations. Access to this interface can either be obtained by typing command directly into the running program, or a whole series of commands can be pre-prepared in an ASCII file and then executed. This 'batch mode' of operation is useful if the same sequence of commands need to be used repeatedly, as for example during the regular program validation exercises.

```
#list 12
full x's u's
end
#sfls
refine
refine
calc
end
#fourier
map type=difference
end
#peaks
end
#distance
end
```

However, for more routine work, it is convenient for the user if frequently used sequences of operations are pre-packed into a single higher-level operation, which is then made accessible from some kind of

Constraints:

SHELXL97

```
EXYZ atomnames  
EADP atomnames
```

CRYSTALS

```
RIDE C(1,X'S) UNTIL F(4)  
RIDE C(1,U'S) UNTIL F(4)  
RIDE C(1,X'S,U'S) UNTIL F(4)  
RIDE C(23,X'S) H(231,X'S) H(232,X'S) H(233,X'S)
```

On a RIDE card (line), parameter shifts are equated on an atom by atom basis

```
LINK C(1,X'S,U'S) UNTIL F(4) AND C(101,X'S,U'S) UNTIL F(104)
```

On a LINK card, parameter shifts in the first atom list (up to the 'AND') are equated with corresponding parameters in the second list. Useful in the initial treatment of pseudo-symmetry or disorder

```
EQUIVALENCE F(1,OCC) UNTIL F(4)
```

A single parameter shift is used for the occupations number of all the atoms between F1 and F4, for example in a BF₄ group

```
EQUIVALENCE F(1,OCC) UNTIL F(4) F(101,OCC) UNTIL F(104)  
WEIGHT -1 F(101) UNTIL F(104)
```

This could be two interpenetrating BF₄ groups (F1-F4 and F101 until F104). Equal shifts are applied to the occupation numbers of all 8 atoms, but in an opposite sense for the second four. A similar construction can be used to impose non-crystallographic symmetry

```
EQUIVALENCE C(1,X) C(101,X)  
WEIGHT -1 C(101,X)
```

A group of atoms can be refined as a rigid body (ie refine their centroid and orientation). A model building instruction must have been used to ensure the group had the correct local geometry before refinement starts.

```
GROUP C(1) UNTIL C(27)
```

SHELXL97 has the option to scale rigid groups (a variable metric) – an option we would certainly look into next time we are extending this part of CRYSTALS.

If a group of parameters are such that their total should remain constant under refinement (e.g. twin components, partial occupancies), this can be applied as a constraint. If one is uncertain of what the total should be, the control can be applied as a restraint with a standard uncertainty. Both options exist in CRYSTALS.

```
SUM parameter list  
e.g. SUM ELEMENT(1) ELEMENT(2) ELEMENT(3) ...  
to ensure that the total of the twin elements add up to unity.  
or SUM s ATOM(1,OCC) ATOM(2,OCC) ATOM(3,OCC) ...  
as a restraint with uncertainty s.
```

Restraints:

SHELXL97

```
DFIX d s[0.02] atom pairs
SADI s[0.02] atom pairs
SAME s1[0.02] s2[0.02] atomnames
DELU/SIMU/ISOR
FLAT s[0.1] four or more atoms
SUMP c sigma c1 m1 c2 m2 ...
```

CRYSTALS

DISTANCE d , s = atom(1) TO atom(2), atom(3) TO atom(4), etc

The distances between the atom pairs are restrained to d with an su of s

DISTANCE d , s = MEAN atom(1) TO atom(2), atom(3) TO atom(4), etc

As above, except that all the pairs are restrained to their common mean $+d$, which is usually zero.

ANGLE a , s = atom₁ TO atom₂ TO atom₃ ... etc

The angle between the 3 atoms is restrained to a degrees with an su of s . A similar syntax enables a number of angles to be restrained to their mean value.

SAME s_1 , s_2 atom list AND atom list AND ...

The geometries at each atom in each list are restrained to be the same. This command is in fact expanded into a list of DISTANCE and ANGLE restraints. s_1 and s_2 are the standard uncertainties on the distances and angles.

VIBRATION u , s = atom₁ TO atom₂, atom₃ TO atom₄, etc

Restrains the component of U along the bond to be the same for each atom in each pair of atoms, (the Hirshfeld condition). u is an offset, generally zero

U(IJ) u , s = atom₁ TO atom₂, atom₃ TO atom₄, etc

The components of U_{ij} for each atom in each atom pair are restrained to be similar, plus an offset, u , usually zero.

PLANAR s atom list

Restrains the atom in the list to be coplanar.

NONBONDED v , p = atom₁ to atom₂, ... etc

The atom pairs are restrained to be v angstrom apart using a function with a steep repulsive gradient if the atoms are too close together, and a shallow attractive gradient if they are further apart.

SUM s parameter list.

Restrains the sum of the listed parameters to remain constant

AVERAGE s parameter list

Restrains each parameter in the list to their common average.

LIMIT s parameter

The shifts in the specified parameters are restrained to zero with a standard uncertainty of s , unless the X-ray data strongly indicate something else. Different parameters or parameter types can be limited to different degrees. This is a derivative of Marquardt's method, and should not be confused with 'damping' a refinement using partial shifts. Partial shifts are also available in CRYSTALS

RESTRAIN v , s = Expression

Expression is a FORTRAN like representation of a function of the refinable parameters and other crystallographic constants, and v is the target value. The expression is differentiated by CRYSTALS and the derivatives added into the normal equations.

CRYSTALS includes functions to go through a structure and automatically generate lists of restraints. These can then either be applied automatically, or manually edited to achieve some non-standard result.

The restraints are found in an ASCII file 'bfile.pch', called the PUNCH file, another hangover from the days when IBM cards were actually punched out!

Automatic generation of restraints is available for:

- Hydrogen atoms. Bond lengths are restrained to the same target values as are used in SHELXL97, bond angles are restrained to target values or to satisfy symmetry, U_{iso} is related to U_{equiv} of the adjacent atoms.

```
#DISTANCE
OUTPUT PUNCH =H-RESTRAIN
END
```

- Create adp similarity and Hirschfeld restraints

```
#DISTANCE
OUTPUT PUNCH = SIMU S1DEV=s S2DEV=u
END
#DISTANCE
OUTPUT PUNCH = DELU D1DEV=s D2DEV=u
END
```

s and u are standard uncertainties for normal and terminal bonded atoms.

- Molecular similarity This was described above (SAME)
- Intermolecular short contacts.

```
#DISTANCE
OUTPUT PUNCH=NON-BONDED VALUE=1.5 POWERFACTOR=1.0
END
```

Non-atomic electron density distributions.

These were described in the last newsletter. For very highly disordered fragments, models based on multiple interpenetrating partially occupied structures become unrealistic. In these cases, it may be better to use a diffuse distribution of electron density with a simple geometric shape, or even to use the discrete Fourier transform of the residual density in the un-resolved area (SQUEEZE). The CRYSTALS integration with SQUEEZE in PLATON conserves the A and B parts of the structure factor, so that no modifications are made to F_o , and the disordered region contributes to the phasing.

*Sandy's examples (Dolomanov *et al.* 2003)*

Example 1. Restraining a disordered BF_4 anion.

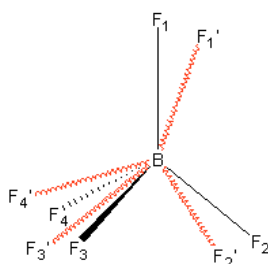


Fig 2: Sandy's figure 7

DFIX	1.38	0.01	B F1	B F2	B F3	B F4		
DFIX	2.25	0.02	F1 F2	F1 F3	F1 F4	F2 F3	F2 F4	F3 F4
DFIX	1.38	0.01	B F1'	B F2'	B F3'	B F4'		
DFIX	2.25	0.02	F1' F2'	F1' F3'	F1' F4'	F2' F3'	F2' F4'	F3' F4'

Becomes

DIST	1.38	0.01	= B(1) to F(1), B(1) to F(2), B(1) to F(3), B(1) to F(4)
CONT			B(1) TO F(101), B(1) TO F(102), B(1) TO F(103), B(1) TO F(104)
DIST	2.25	0.02	= F(1) to F(2), F(1) to F(3), F(1) to F(4),
CONT			F(2) to F(3), F(2) to F(4), F(3) to F(4)
CONT			F(101) TO F(102), F(101) TO F(103), F(101) TO F(104) etc

Note that atoms are identified by an element type and a number. The advantage of this is that the numbers can be used in mathematical expressions, eg for sorting, in CRYSTALS.

Example 2. Restraining a poorly defined SbF_6^- anion. The target value for the Sb-F bond is uncertain, but they are expected to be the same by symmetry.

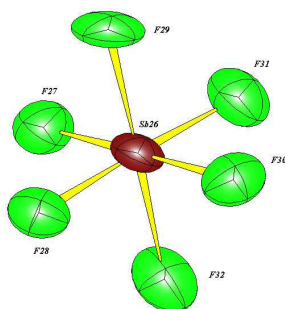


Fig 3: Sandy's figure 8

SADI	0.01	Sb26	F27	Sb26	F28	...	Sb26	F32
------	------	------	-----	------	-----	-----	------	-----

Becomes

DIST	0.0, .01	= MEAN SB(26) TO F(27), SB(26) TO F(28) ..SB(26) TO F(32)
------	----------	---

Example 3. Restrain the local geometries of three fragments to be similar.

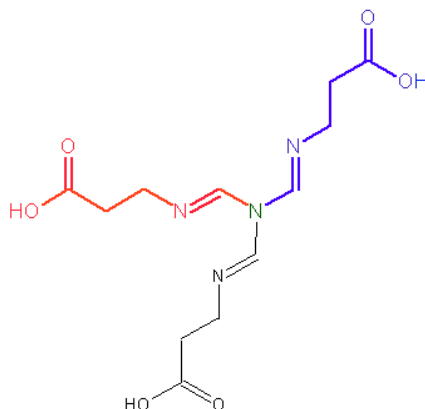


Fig 4: Sandy's figure 9

```

SAME 0.01 C1 > O7
C11 ...
N12 ...
C13 ...
C14 ...
C15 ...
O16 ...
O17 ...

SAME 0.01 C1 > O7
C21 ...
N22 ...
C23 ...
C24 ...
C25 ...
O26 ...
O27 ...

```

Becomes

```

SAME .01 C(1) UNTIL O(7) AND C(11) UNTIL O(17) AND C(21) UNTIL O(27)

```

Example 4. A disordered bromide anion in a cavity.

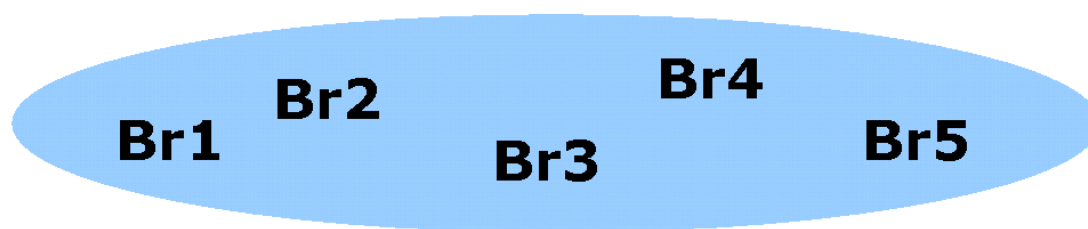


Fig 5: *Sandy's figure 10*

SUMP	1.00	0.01	1	2	1	3	1	4	1	5	1	6
FVAR	osf	0.2	0.2	0.2	0.2	0.2						
Br1	5	x	y	z	21							
Br2	5	x	y	z	31							
Br3	5	x	y	z	41							
Br4	5	x	y	z	51							
Br5	5	x	y	z	61							

Becomes

As a constraint:

```

SUM BR(1,OCC) UNTIL BR(5)

```

Or as a restraint

```

SUM .01 BR(1,OCC) UNTIL BR(5)

```

It is assumed that the Br are adjacent in the atom list, and that the initial sum of their occupancy factors adds up to unity. If any of the Br also happen to be on a special position, more complex restraints must be used.

Free Variables

Sandy's last example illustrates the use of 'Free Variables' in SHELXL97. Free variables are also used in CRYSTALS, but they are automatically set up for the user.

Blocked Normal Matrices

In SHELXL97 the BLOC instruction enables the user to specify that different groups of parameters can be processed in different ways in different least squares cycles. Two different strategies are available in CRYSTALS.

Multi-block sparse matrix. User-specified blocks of parameters are set up along the leading diagonal. The most sparse is to use a single block for each atom. More effective strategies are to set up blocks either corresponding to molecular fragments, or to separate the positional parameters and the adps. There are no restrictions on the blocking scheme other than that a given parameter can only appear in one block. The figure shows an atom-by-atom blocking scheme, and a per-fragment scheme.

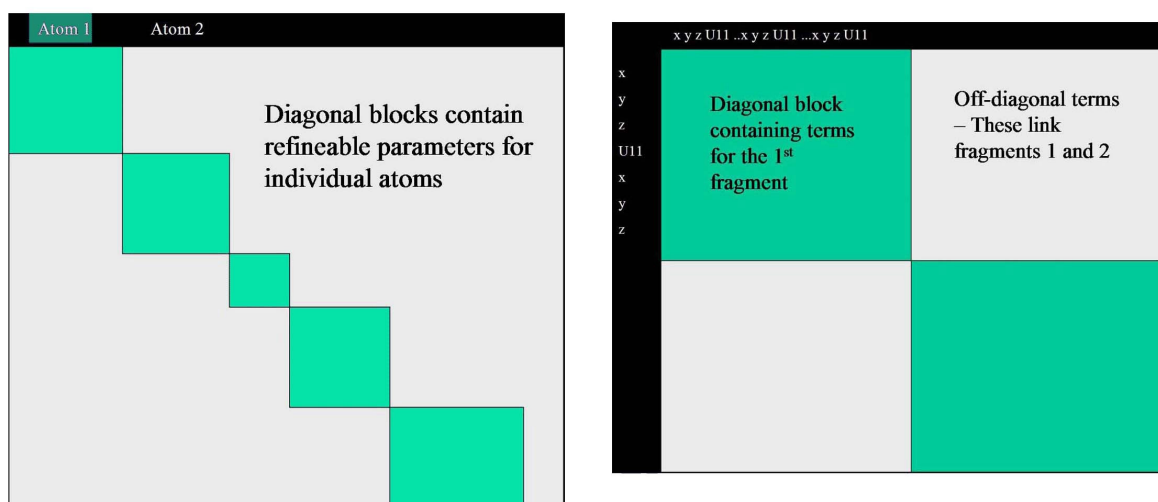


Fig 6: Sparse Matrix Schematics. In the left example, a block of the matrix is computed for the selected parameters of the selected atoms. Cross terms between atoms are ignored. In the right example a separate block is set up for each fragment. This is an excellent scheme if the fragments are not correlated, but is a catastrophe if they are related by pseudo symmetry. Beware of being Marshded! (Watkin, 1994)

```
BLOCK X' S
BLOCK SCALE FIRST(U'S) UNTIL C(27) C(28,U[ISO]) UNTIL LAST
...
REFINE
REFINE
REFINE
```

The first two lines create a matrix consisting of two blocks. One block contains the x.y & z of all the atoms. The second block contains the anisotropic adps of the atoms up to C28, and the isotropic adps of the rest. Note that this block also includes the overall scalefactor, which is highly correlated with the adps. This sparse matrix is accumulated and inverted for all three cycles of refinement

1) Small dense matrices accumulated and inverted separately in different cycles

```
BLOCK X' S  
REFINE  
BLOCK SCALE FIRST(U' S) UNTIL C(27) C(28,U[ISO]) UNTIL LAST  
REFINE
```

This task consists of two independent cycles of refinement. In the first only the positions are refined, in the second the scale and adps.

The first example is the preferred way of completing the refinement of a very large structure since a single structure factor calculation is used for the computation of all the derivatives.

Model Building

This is perhaps the area of greatest dissimilarity between SHELXL97 and CRYSTALS. In SHELXL97 the modelling instructions are embedded amongst the constraints and restraints. In CRYSTALS, modelling, constraints and restraints are kept quite distinct. As usual, this gives SHELXL97 an advantage in terms of lines typed. However, because CRYSTALS is generally run interactively on a PC, separating the functions enables the user to try various models in an exploratory way. A difficult analysis can be put aside and then picked up at a later date at the point where it was previously left off – in much the same way as a complex document can be worked on in Microsoft WORD without having to restart from a plain text file.

In the last example (4), both the restraint and the constraint instructions assume that the starting model has been correctly built, and so refer only the actual parameter values to be refined. The same idea is used for the treatment of hydrogen atoms. Firstly they are added to the backbone (either from a difference map or geometrically), and then their refinement is controlled (by fixing them, by applying restraints, by applying 'riding' constraints or by any appropriate mixture).

CRYSTALS:

```
#PERHYDRO  
DIST 0.98  
U[ISO] NEXT 1.2  
END
```

This command places hydrogen atoms on all the carbon atoms in the structure at a distance of 0.98 Angstrom, and with U_{iso} equal to 1.2 times the carbon U_{equiv} .

Commands also exist for adding hydrogen atoms on a one-by-one basis to selected atoms. In fact, the 'PERHYDRO' command is expanded into a series of per-atom commands, which the user can view and edit to produce special environments.

```
#HYDROGEN  
DIST 1.00  
U[ISO] NEXT 1.1  
H13 pivot_atom 3 neighbouring_atoms  
H12 pivot_atom 2 neighbouring_atoms  
etc  
HBOND donor acceptor  
END
```

Example 5. Adding in part of a structural model from elsewhere.

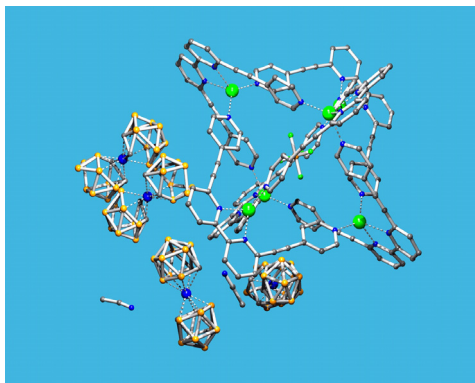


Fig 7: Sandy's figure 4

SHELXL97.

```
FRAG 17 15.72 20.15 20.39 74.8 70.75 86.50
Co 4 x y z...
C1 1 x y z...
C2 1 x y z...
B3 3 x y z...
...
B19 3 x y z...
FEND

AFIX 17
Co1 7 0.33250 0.76245 0.52909 11.000 0.0608 0.1389 =
      0.0396 -0.0183 -0.0212 0.0153
C1 1 0.37668 0.84367 0.54903 11.00000 0.155
C2 1 0.41382 0.84350 0.45796 11.00000 0.089
B3 3 0.31680 0.82455 0.43612 11.00000 0.117
...
B19 3 0.20793 0.79538 0.51437 11.00000 0.138
AFIX 0
```

CRYSTALS

The model, which is assumed to have approximate values for most atoms, is used as the target for regularisation by a better structure. This may either be a well defined group within the current structure, or come from elsewhere. In this case atomic coordinates for the ideal structure come from the CSD.

```
#REGULARISE REPLACE
GROUP 17
SYSTEM 15.72 20.15 20.39 74.8 70.75 86.50
IDEAL
ATOM X Y Z
...
ATOM X Y Z
TARGET CO(1) UNTIL B(19)
END
```

There are 17 ATOM lines containing the coordinates of the atoms in the same order as approximate ones in the group Co(1) until B(19).

Refinement is completed by refining the atoms freely, with geometric restraints or as a rigid group:

```
GROUP CO(1) UNTIL B(19)
```

The REGULARISE command can also be used to fill in missing atoms in one fragment with ones mapped from elsewhere (REGULARISE AUGMENT), to compare two fragments (ie find the rms atomic deviations after fitting them together with information about the transformation used) or to replace a trial model with a scaleable geometric figure (CP-RING, HEXAGON, OCTAHEDRON, PHENYL, SQUARE, TBP, TETRAHEDRON).

Residues and Parts.

SHELXL97 included a powerful syntax for dealing with multi-fragment or disordered structures. Similar ideas have been implemented in CRYSTALS. Groups of atoms which are spatially distinct (main molecule, solvent, counter ions) can each be put into a separate '*residue*'. These residues can be used in a command anywhere an atom identifier might be used as a short hand for a whole group of atoms.

SHELXL97

```
PART n sof
```

CRYSTALS

```
FULL FRAG (1, OCC)
```

Might be equivalent to:

```
FULL C (1, OCC) UNTIL C (27)
```

Disordered parts of a molecule (or fragment) can be distinguished from each other by assigning them to a '*group*'. Atoms in different groups do not bond to each other, unless the group number is zero. This can be useful for setting up constraints.

```
RIDE GROUP (1, OCC) AND GROUP (2, OCC)
```

The definition of group 1 and 2 is part of building the model.

Summary

This part of this article has tried to show that there are great similarities between SHELXL97 and the command line interface to CRYSTALS. This is the interface used by experienced crystallographers when dealing with difficult problems. To aid the user in composing correct commands, a view of the model is always available on the screen, and atom names can be passed into the command line simply by clicking on them in the diagram. This is most useful during model building – and if the model begins to look 'wrong', the user can regress to an earlier model.

CRYSTALS and SIR2004

The SIR family of programs are widely respected and widely used because of their direct methods capabilities. However, since *SIR97* they have also included refinement code developed from CAOS. Unlike the close similarity between CRYSTALS & SHELXL97 (an example of convergent evolution), the close similarity between CAOS and CRYSTALS is the result of shared ancestry. Much of the underlying data base design in CAOS was devised by Bob Carruthers during his post doctoral stay in Rome, before returning to Oxford to begin writing CRYSTALS. The design of this data structure was strongly influenced by earlier ALGOL programs of Durward Cruickshank and AUTOCODE programs of John Rollett. The similarities include a *list structure* for maintaining the data, and an input syntax amenable to syntactical processing.

The constraints and restraints available in CRYSTALS have been described above. The recent article on *SIR2004* lists some of the refinement features:

- 1) *The ability to reduce the full matrix of the normal equations defining any kind of blocks.*
This facility is available in CRYSTALS, but is rarely used now except for the biggest structures. CRYSTALS will handle 600 anisotropic or 1400 isotropic atoms in the full matrix, and an effectively unlimited number with matrix blocking. However, the algorithms used to compute structure factors and derivatives are not optimal for macromolecular structures. In addition to the normal Choleski inverter, CRYSTALS can solve the normal equations by eigenvalue decomposition.
- 2) *18 weighting schemes are available.*
CRYSTALS also contains most of the schemes available in SIR2004, plus a few others. Robust-resistant and Dunitz Seiler weight modifiers are available in conjunction with any of the other schemes. A re-implementation of the SHELXL97 weighting scheme is suitable for F^2 refinement.
- 3) *The program generates constraints for the parameters of atoms on special positions in all space groups.*
CRYSTALS tries to generate space group constraints for all atomic parameters. However, there are difficulties in symbolically reconciling space group constraints when the user has already manually specified other constraints (eg equivalencing of parameter shifts, treating a group of atoms as a rigid body, reparameterisation into orthogonal coordinates). In this case, space group symmetry requirements are applied to the affected parameters in the form of very tight restraints.
- 4) *Automatic or through wizard generation of hydrogen atoms.*
In CRYSTALS the command PERHYDRO geometrically places hydrogen atoms on all carbon atoms, otherwise the menu shown below can be used for individual cases. A semi-automated procedure combines Fourier and geometric methods.
- 5) *The possibility to impose conditions (constraints) or additional information (restraints).*
Many of the options available in CRYSTALS were described in the comparison with SHELXL97. An additional constraint, useful when there is high correlation between parameters, is to refine sums and differences of parameter shifts, rather than the shifts themselves

```
COMBINE C(1,X'S) UNTIL C(6) AND C(101,X'S) UNTIL C(106)
```

This would re-parameterise two six-membered groups related by a pseudo crystallographic inversion centre.

- 6) *Floating origin is restrained automatically by setting the restrain of the sum of the. appropriate coordinate.*
The terms in the sum are weighted according to the scattering power of the atoms.
- 7) *Refinement of the Flack parameter to evaluate the absolute configuration.*
This is performed correctly, including the correlation with any other refined parameters. The parameter is permitted to take on negative values. Other refinement features available in CRYSTALS include twin, batch (for multi-crystal data sets) and layer scale factors.

The underlying codes in both SIR2004 and CRYSTALS are almost equally suitable for most routine structure refinements. There are however significant differences in the user interface.

Both SIR2004 and CRYSTALS were originally controlled by cards or card images in ASCII files. Development has continued along parallel but independent routes, and both programs now have graphical interfaces. The figure shows how close these developments sometimes are.

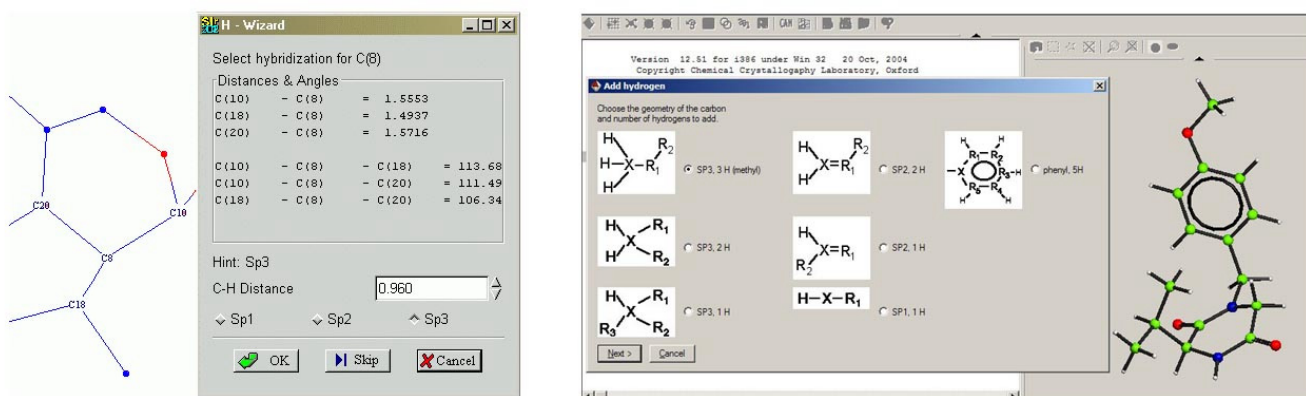


Fig 8: Parallel evolution in SIR2004 and CRYSTALS. Hydrogen placement menus: left, CAOS, right CRYSTALS.

One very special feature of CRYSTALS is the integral processor for a tailored scripting language. This enables a user or application programmer to construct quite complicated operations out of a sequence of CRYSTALS primitives. For example, the primitive for placing hydrogen atoms on carbon can be combined with the primitive for computing a difference electron density map. The results are displayed jointly for possible user intervention, after which the refinable model is updated. One option is to perform automatic Fourier refinement, and another is to refine the (found or placed) hydrogen atoms with automatically generated slack restraints on the geometry and isotropic adps. Once the geometry has been regularised in this way, the refinement can be finished with the hydrogen atoms treated with riding constraints. These also are generated automatically.

The figure shows the structure manipulation menu and the hydrogen treatment menu.

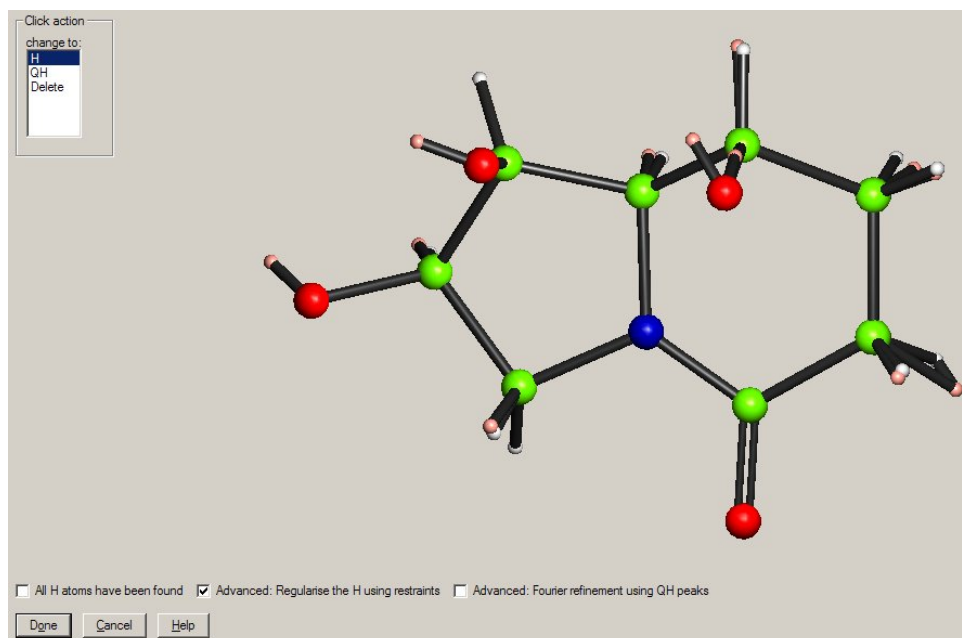


Fig 9: Hydrogen placing. Selections can be made in the model by clicking. The pink atoms are peaks from the difference density map, the white atoms are hydrogen placed geometrically. The user can change the peaks found in the difference map into hydrogen atoms. The check-box near the middle will initiate cycles of refinement of the hydrogen atoms only with the imposition of suitable geometric and adp restraints.

Structure Editing.

The screenshot displays the COOT software interface. The 'Structure' menu is open, showing options such as 'Undo/backup model', 'Input', 'Edit co-ordinates', 'Invert co-ordinates', 'Change types of atoms', 'Renummer atoms', 'Collect atoms by symmetry', 'Distances', 'Add hydrogen geometrically', 'Add hydrogen + fourier', 'Add hydrogen manually', 'Remove hydrogen', 'Remove Fourier peaks', and 'Help/Information'. A context menu for atom C(9) is also visible, listing actions like 'Delete C(9)', 'Environment of C(9)', 'Select fragment containing C(9)', 'Zoom fragment containing C(9)', 'Delete group', 'Centroid of group', 'Slant Fourier map', 'Voids map', 'TLS analysis', 'Bonding', 'Geometry', 'Restraints', 'Sticky refine mode', and 'Change types to...'. The background shows a 3D molecular model of a protein-ligand complex.

The actual numerical values of parameters can be changed by using a text editor on an ASCII file containing all the parameters for the whole model. More conveniently, some can be changed using the drop down context sensitive menu. Alternatively, an editor pane can be opened for any atom and parameter values changed interactively.

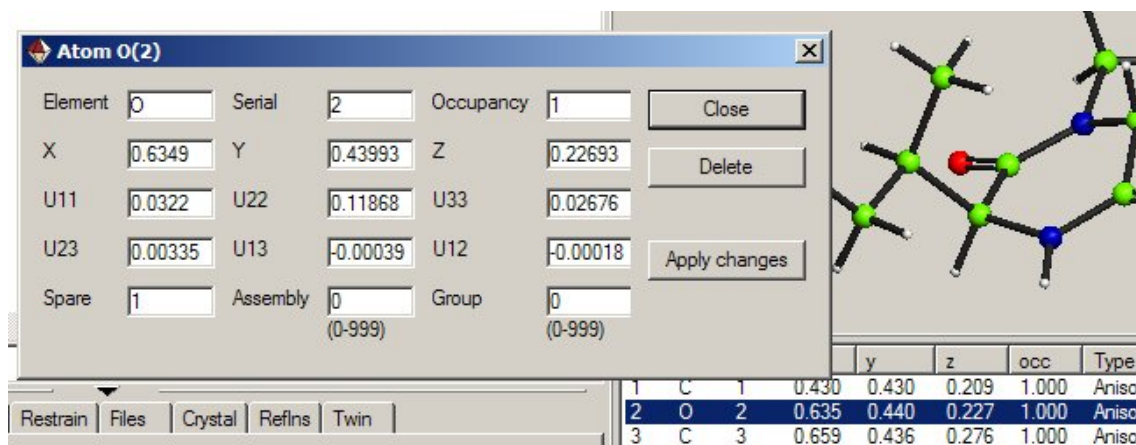


Fig 11: *GUI menu of modelling primitives*

The greatest wealth of modelling primitives is accessible via the old-fashioned command-line input. This is still available via a text input box in the GUI. Users can either type commands into this box, or enter the name of a file of preprepared commands. There is a vast vocabulary of commands that can be entered here. The built-in editor enables simple modification of parameter values, but more useful are crystallographic operations. These can perform mathematical operations on groups of atom parameters, change, sort and filter atoms, assign atoms to residues or groups – a total of 40 different operations.

Modelling Examples

1. Move the centre of gravity of a structure onto a centre of inversion.
- 2.

```
#EDIT
CENTROID 100 ALL
SHIFT -x -y -z ALL
SHIFT .5 .5 .5 ALL
DELETE QC(100)
END
```

This creates a pseudo atom (QC(100)) at the centroid of the existing atoms. The model is then shifted by the coordinates of this centroid (-x,-y,-z) to put the centre of gravity at (0,0,0), and then shifted again to put it at (.5,.5,.5). The pseudo atom is then deleted. Useful if a $P\bar{1}$ structure has been solved in $P\bar{1}$.

3. Rotate a methyl group through 30 degrees.
- 4.

```
#EDIT
ROTATE 60.0 C(1) C(2) H(20) H(21) H(22)
END
```

5. Allocate residue numbers to discrete moieties, add an offset to the serial numbers of moiety two, and delete moiety three.

```
#EDIT
INSERT RESIDUE
ADD 100 RESIDUE(2, SERIAL)
DELET RESIDUE(3)
END
```

6. Reject any atoms whose isotropic adp has become too large

```
#EDIT
SELECT U[ISO] LE .10
END
```

7. Ensure that the individual adps of a group conform to a rigid model before starting restrained refinement

```
#ANISO
ATOM C(1) UNTIL C(6)
TLS
REPLACE
END
```

This computes a tls model for the listed atoms, and then replaces the individual Uaniso with values computed from the tls model. Elements of the tls model can be changed manually if required. The tls model can be extended to atoms not in the original calculation.

8. After the space group has been changed from P 1 to P $\bar{1}$, one half of the structure can simply be deleted, or the two halves can be averaged.
- 9.

```
#PEAK 5 5
SELECT TYPE=AVERAGE
END
```

If a structure has $Z' > 1$, it is convenient if the atoms are numbered in a consistent way. There is a SCRIPT to help with this. The user numbers one of the molecules tidily, then propagates this numbering into the other molecules. If the atoms in the other molecules are of type Q (ie un-named peaks) the atom typing is also propagated.

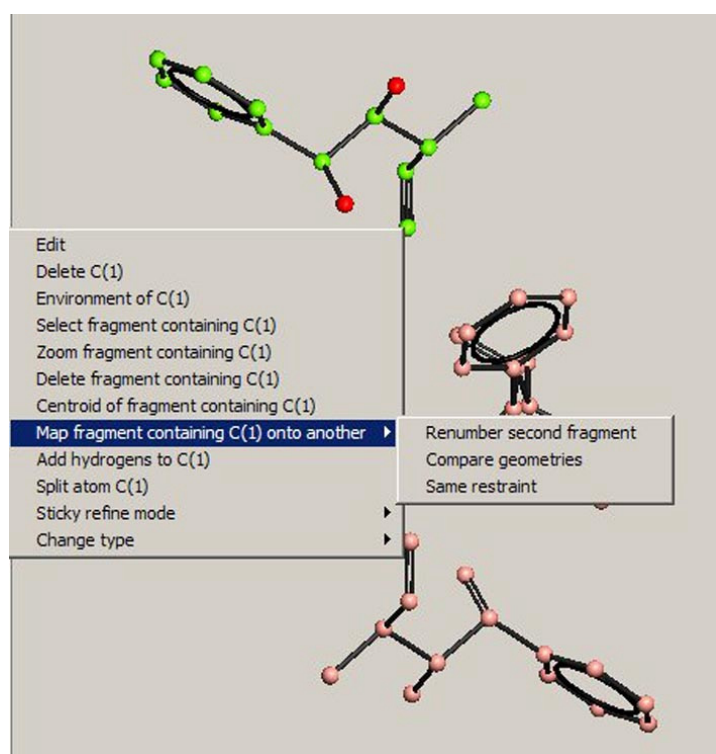


Fig 12: The numbering and atom typing for the top molecule can be propagated into the other molecules.

After this, the molecules can be restrained to be similar

```
SAME C(1) UNTIL O(23) AND C(101) UNTIL O(123) etc
```

Validation

Refinement programs rarely ‘blow up’ these days, so that it is possible for a user to generate an apparently stable yet incorrect model. Various tools to help with validation are available in CRYSTALS. In the ‘Guided’ mode of operation (not so far mentioned in this article), the user is guided through the analysis step by step. The penultimate stage consists of validating the analysis against the minimal Acta Cryst criteria. Beyond this, if the user has access to PLATON or MOGUL, tasks can be spawned to these programs and the results acted upon in CRYSTALS. For more troublesome case, there are tools for looking into the data:

1. Rotax. The original Edinburgh code has been brought right into CRYSTALS. If twinning is detected, the appropriate twin laws can be applied directly and refinement continued.

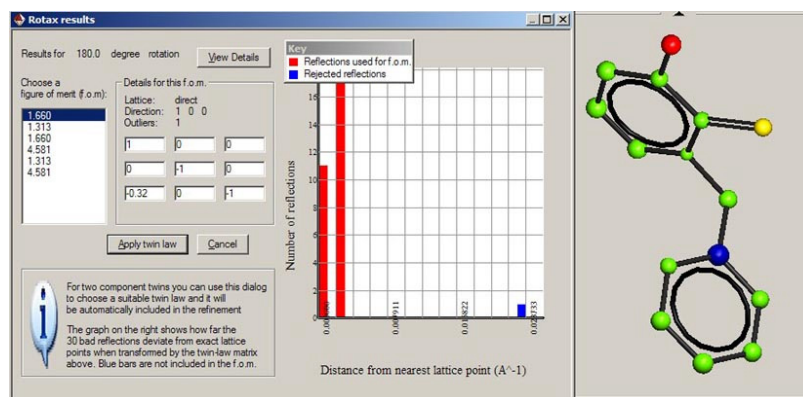


Fig 13: GUI interface for Rotax inside the Crystals software

2. Examination of the original data. A weird Wilson Plot, or unexpected incompleteness of data may indicate something went wrong during data collection.

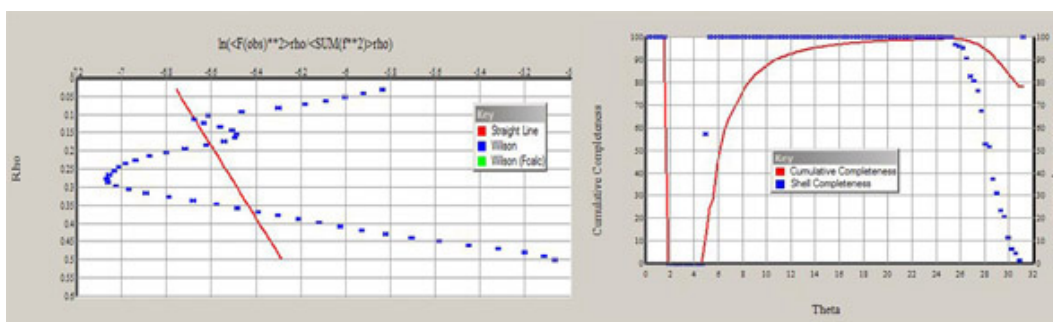


Fig 14: Wilson plot (left) shows something seriously wrong with the high angle data. The completeness chart (right) tends to confirm this.

3. If refinement gets bogged down, look for trends in F_o and F_c . A common source of difficulties is partially occluded low angle strong reflections. They have a minimal impact on direct methods, but can be a catastrophe for refinement. Outliers on an F_o - F_c plot may either be flawed data, or indicative of something missing from the model.

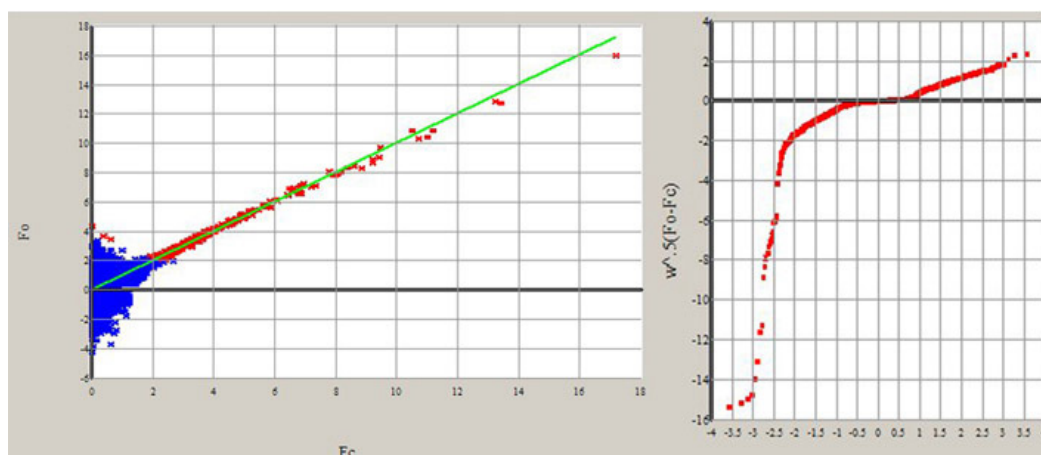


Fig 15: The F_o - F_c plot (left) shows no real outliers, but masses of data with small or negative F_o values. The normal probability plot (right) also confirms that there is something seriously wrong with the model or the data.

4. In the case illustrated above (2 and 3), the data had been collected at break-neck speed on a Nonius Kccd diffractometer. Of the 4143 reflections recorded, 2848 were greater than zero, and only 564 (14%) had $I > 3 \sigma(I)$. Even so, chemically and statistically acceptable models were achieved without the use of restraints. Note that even with such a weak data set, the structure was solved by *SIR92* without serious difficulty.

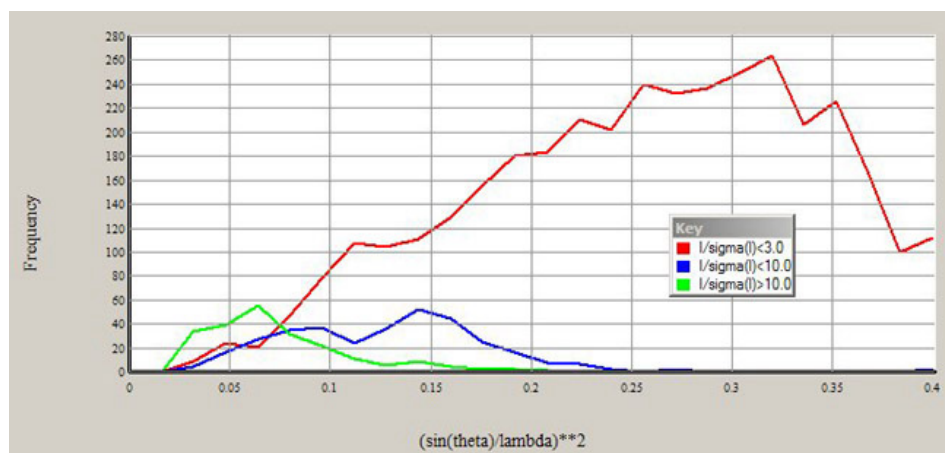


Fig 16: Cumulative distribution curves for the very weak data (red), middling (blue) and strong data (green) plotted as a function of resolution. 86% of the data has $I < 3 \sigma(I)$. No one except a programmer would choose to collect such weak data, but sometimes Nature offers no choice.

Conclusion.

SHELXL97, SIR2002 and CRYSTALS all have a great wealth of mathematical and crystallographic resources in common. The differences begin to emerge when large or non-routine problems occur. In these cases the crystallographer needs both the best available mathematical methods, and a helpful environment for applying the techniques. We feel that the ability to interactively make changes to the model, and instantly apply a complex refinement strategy is particularly useful in these cases. The validity of a particular model and strategy can be tested against a wealth of yard-sticks.

The complete CRYSTALS package, including GUI, manuals, examples and a high quality graphics module, is available at no charge for non-commercial use from <http://www.xtl.ox.ac.uk>.

References.

O.V. Dolomanov, A.J. Blake, N.R. Champness, M. Schröder & C. Wilson, *Chemical Communications* 2003. "A novel synthetic strategy for hexanuclear supramolecular architectures", pp. 682–683

D.J. Watkin, *Acta Cryst.* 1994, **A50**, 411–437

cctbx news: Phil and friends

Ralf W. Grosse-Kunstleve, Pavel V. Afonine, Nicholas K. Sauter and Paul D. Adams,
*Computational Crystallography Initiative, Lawrence Berkeley National Laboratory, 1 Cyclotron Road,
BLDG 64R0121, Berkeley, California 94720-8118., USA - Email : RWGrosse-Kunstleve@lbl.gov ;
WWW: <http://cci.lbl.gov/>*

Abstract

We describe recent developments of the Computational Crystallography Toolbox.

Preamble

In order to interactively run the examples scripts shown below, the reader is highly encouraged to visit http://cci.lbl.gov/cctbx_build/ and to download one of the completely self-contained, self-extracting binary *cctbx* distributions (supported platforms include Linux, Mac OS X, Windows, IRIX, and Tru64 Unix). All example scripts shown below were tested with *cctbx* build 2005_01_22_0855.

In the following we refer to our articles in the previous issues of this newsletter as "Newsletter No. 1", "Newsletter No. 2", etc. to improve readability. The full citations are included in the references section.

1 Introduction

The *Computational Crystallography Toolbox* (*cctbx*, <http://cctbx.sourceforge.net/>) is the open-source component of the Phenix project (<http://www.phenix-online.org/>). Currently much energy is devoted to implementing a streamlined command-line interface to the Phenix refinement algorithms. In this article we describe the new *Python-based hierarchical interchange language* (Phil) that was developed for this purpose. Other important developments highlighted below are our implementation of cartesian dynamics simulated annealing for macromolecular structure refinement, the significant enhancements of the `iotbx.reflections_statistics` command, the new C++ and Python interfaces to the CCP4 MTZ library, and the inclusion of PyCifRW in the *cctbx* bundles available for download.

The command-line interface to the Phenix refinement algorithms is called `phenix.refine`. The refinement algorithms require a structural model, xray data and optionally experimental phase information, typically in the form of Hendrickson-Lattman coefficients. For macromolecular refinement the ratio of experimental observations to refinable parameters is typically quite low. Geometry restraints have to be included in order to make the refinement stable. Finally, the refinement algorithms introduce a large number of parameters, such as the number of refinement cycles to run, parameters for bulk-solvent correction, simulated annealing, etc. In our current development version the number of parameters including file names and data labels is already greater than 100. This number is likely to increase significantly as we add more features in the future.

In previous issues of this newsletter we have described comprehensive utilities for reading reflection files (Newsletter No. 3), processing of structural data formatted as PDB files integrated with the handling of geometry restraints based on the CCP4 Monomer Library (Newsletter No. 4). However, until recently we had only ad-hoc solutions for the handling of the large number of algorithmic parameters. `phenix.refine` is written in Python (with C++ extensions for numerically intensive algorithms, see Newsletter No. 1). Therefore it was quite natural for us to also use Python to define parameters. For example, Python classes are quite convenient for organizing parameters:


```

from libtbx import introspection

class cartesian_dynamics:
    def __init__(self, temperature      = 300,
                  number_of_steps     = 200,
                  time_step            = 0.0005):
        introspection.adopt_init_args()

class simulated_annealing:
    def __init__(self, do_simulated_annealing = False,
                  start_temperature          = 2500,
                  final_temperature          = 300,
                  cool_rate                  = 25,
                  number_of_steps            = 25,
                  time_step                  = 0.0005,
                  update_grads_shift         = 0.3):
        introspection.adopt_init_args()

```

A group of parameters can then be used like this:

```

my_cartesian_dynamics_params = cartesian_dynamics(number_of_steps=300)
my_simulated_annealing_params = simulated_annealing(final_temperature=200)
some_algorithm(
    cartesian_dynamics_params=my_cartesian_dynamics_params,
    simulated_annealing_params=my_simulated_annealing_params)

```

With:

```

def some_algorithm(
    cartesian_dynamics_params,
    simulated_annealing_params):
    print cartesian_dynamics_params.temperature
    print cartesian_dynamics_params.number_of_steps
    print simulated_annealing_params.start_temperature
    print simulated_annealing_params.final_temperature

```

the output is:

```

300
300
2500
200

```

This shows that we retain the default values for `temperature` and `start_temperature`, but override the values for `number_of_steps` and `final_temperature`.

2 Management of parameters: Phil is your friend

One obvious problem of the approach to parameter management outlined above is that it requires familiarity with the Python syntax. While Python is arguably one of the most elegant programming languages, it still has too much syntax for non-programmers. E.g. all Python string literals have to be in quotes and indentation is syntactically significant. It also appeared difficult to implement the advanced parameter management features introduced below working exclusively with Python syntax. Therefore we

have replaced the Python syntax with the new Phil syntax to make parameter management as simple as possible. The Phil equivalent of the examples above is:

```
refinement.cartesian_dynamics {  
  temperature = 300  
  number_of_steps = 200  
  time_step = 0.0005  
}  
  
refinement.simulated_annealing {  
  do_simulated_annealing = False  
  start_temperature = 2500  
  final_temperature = 300  
  cool_rate = 25  
  number_of_steps = 25  
  time_step = 0.0005  
  update_grads_shift = 0.3  
}
```

The Phil syntax has only two main elements, `phil.definition` (e.g. `cool_rate=25` and `phil.scope` (e.g. `simulated_annealing { }`). To make this syntax as user-friendly as possible, strings do not have to be quoted and, unlike Python, indentation is not syntactically significant. E.g. this:

```
refinement.xray_data {  
  file_name = "peak.mtz"  
  labels = "Fobs" "SigFobs"  
}
```

is equivalent to:

```
refinement.xray_data {  
file_name=peak.mtz  
labels=Fobs SigFobs  
}
```

Scopes can be nested recursively. The number of nesting levels is limited only by Python's recursion limit (default 1000). To maximize convenience, nested scopes can be defined in two equivalent ways. For example:

```
refinement {  
  xray_data {  
  }  
}
```

is equivalent to:

```
refinement.xray_data {  
}
```

2.1 Beyond syntax

Phil is more than just a parser for a very simple, user-friendly syntax. Major Phil features are:

- The concepts of *master files* and *user files*. The syntax for the two types of Phil files is identical, but the processed Phil files are used in different ways. I.e. the concepts exist only at the semantical level. The "look and feel" of the files is uniform.
- Interpretation of command-line arguments as Phil definitions.
- Merging of (multiple) Phil files and (multiple) Phil definitions derived from command-line arguments.
- Automatic conversion of Phil files to pure Python objects equivalent to instances of ad-hoc Python parameter classes like the examples shown in the introduction. These pure Python objects are completely independent from the Phil system.
- The reverse conversion of (potentially modified) pure Python objects back to Phil files. This could also be viewed as a Phil pretty printer.
- Shell-like variable substitution using \$var and \${var} syntax.
- `include` syntax to merge Phil files at the parser level.

2.2 Master files

The master files are written by the software developer and include "attributes" for each parameter, such as the type (integer, floating-point, string, unit cell, etc.) and support information for graphical interfaces. For example:

```
refinement.crystal_symmetry {  
    unit_cell=None  
        .type=unit_cell  
        .help="Unit cell parameters."  
        .input_size = 40  
        .expert_level = 0  
    space_group=None  
        .type=space_group  
        .help="Space group symbol or number."  
        .input_size = 20  
        .expert_level = 0  
}
```

To see the full set of "attributes" for all `phenix.refine` parameters run this command:

```
iotbx.phil --show_all_attributes $MMTBX_DIST/mmtbx/refinement/__init__.params
```

The output is not shown because it is more than 1000 lines long (and still growing). Fortunately, the end-user does not have to be aware of these long master files.

2.3 User files

User files are typically generated by the application, e.g.

```
phenix.refine --show_defaults
```

will process the master file. (Since phenix is not open source this command is not available in a plain *cctbx* installation.) This command will list only the most relevant parameters, classified by the software developer as `.expert_level = 0`. For example:

```
refinement.crystal_symmetry {  
    unit_cell = None  
    space_group = None  
}
```

The attributes are not shown. Therefore the output is much shorter compared to the `iotbx.phil` output above. Currently the output contains only 53 lines with 35 definitions.

2.4 Command-line arguments + Phil

In theory the user could save and edit the generated parameter files. However, in most practical situations this is not necessary for two reasons.

Firstly, `phenix.refine` (and in the future other *cctbx* and Phenix applications) inspects all input files and uses the information found to fill in the blanks automatically. For example the unit cell is copied from the input PDB file or, if this information is missing in the PDB file, from a reflection file. This is not only convenient, but also eliminates the possibility of typing errors.

Secondly, command-line arguments that are not file names or options prefixed with `--` (like `--show_defaults` above) are given to Phil for examination. E.g., this is a possible command:

```
phenix.refine peak.mtz model.pdb number_of_macro_cycles=10
```

Assume the first two arguments can be opened as files (the file names may be specified in any order; `phenix.refine` detects the file types automatically). Also assume that a file with the name `number_of_macro_cycles=10` does not exist. This argument will therefore be interpreted with Phil.

2.5 Merging of Phil objects

The Phil parser converts master files, user files and command line arguments to uniform Phil objects which can be merged to generate a combined set of "effective" parameters used in running the application. We demonstrate this by way of a simple, self-contained Python script with embedded Phil syntax:

```
import iotbx.phil  
  
master_params = iotbx.phil.parse("""  
    refinement.crystal_symmetry {  
        unit_cell = None  
        .type=unit_cell
```

```

    space_group = None
    .type=space_group
}
""")

user_params = iotbx.phil.parse("""
  refinement.crystal_symmetry {
    unit_cell = 10 12 12 90 90 120
    space_group = None
  }
""")

command_line_params = iotbx.phil.parse(
  "refinement.crystal_symmetry.space_group=19")

effective_params = master_params.fetch(
  sources=[user_params, command_line_params])
effective_params.show()

```

The `master_params` define all available parameters including the type information. The `user_params` override the default `unit_cell` assignment but leave the space group undefined. The space group symbol is defined by the command line argument. `effective_params.show()` produces:

```

refinement.crystal_symmetry {
  unit_cell = 10 12 12 90 90 120
  space_group = 19
}

```

Having to type in fully qualified parameter names (e.g. `refinement.crystal_symmetry.space_group`) can be very inconvenient. Therefore Phil includes support for matching parameter names of command-line arguments as substrings to the parameter names in the master files:

```

import libtbx.phil.command_line

argument_interpreter = libtbx.phil.command_line.argument_interpreter(
  master_params=master_params,
  home_scope="refinement")

command_line_params = argument_interpreter.process(
  arg="space_group=19")

```

This works even if the user writes just `group=19` or even `e_gr=19`. The only requirement is that the substring leads to a unique match in the master file. Otherwise Phil produces a helpful error message. For example:

```

argument_interpreter.process("u=19")

```

leads to:

```

UserError: Ambiguous parameter definition: u = 19
Best matches:
  refinement.crystal_symmetry.unit_cell
  refinement.crystal_symmetry.space_group

```

The user can cut-and-paste the desired parameter to edit the command line for another trial to run the application.

2.6 Conversion of Phil objects to pure Python objects

The Phil parser produces objects that preserve most information generated in the parsing process, such as line numbers and parameter attributes. While this information is very useful for pretty printing (e.g. to archive effective parameters) and the automatic generation of graphical user interfaces, it is only a burden in the context of core numerical algorithms. Therefore Phil supports "extraction" of light-weight pure Python objects from the Phil objects. Based on the example above, this can be achieved with just one line:

```
params = effective_params.extract()
```

We can now use the extracted objects in the context of Python:

```
print params.refinement.crystal_symmetry.unit_cell
print params.refinement.crystal_symmetry.space_group
```

Output:

```
(10, 12, 12, 90, 90, 120)
P 21 21 21
```

At first glance one may almost miss that something significant has happened. However, we started out with "space_group=19" and now we see P 21 21 21 in the output. This is because the space_group parameter was defined to be of .type=space_group in the master file. Associated with each type are converters to and from corresponding Python objects. In this case, the space_group converter produces a Python object of type:

```
print repr(params.refinement.crystal_symmetry.space_group)
```

Output:

```
<cctbx.sgtbx.space_group_info instance at 0xb64edf6c>
```

This object cannot only show the space group symbol, but has many other "methods". E.g. to print the list of symmetry operations in "xyz" notation:

```
for s in params.refinement.crystal_symmetry.space_group.group():
    print s
```


Output:

```
x, y, z
x+1/2, -y+1/2, -z
-x, y+1/2, -z+1/2
-x+1/2, -y, z+1/2
```

2.7 Conversion of Python objects to Phil objects

Phil also supports the reverse conversion compared to the previous section, from Python objects to Phil objects. For example, to change the unit cell parameters:

```
from cctbx import uctbx

params.refinement.crystal_symmetry.unit_cell = uctbx.unit_cell(
    (10, 12, 15, 90, 90, 90))
modified_params = master_params.format(python_object=params)
modified_params.show()
```

Output:

```
refinement.crystal_symmetry {
  unit_cell = 10 12 15 90 90 90
  space_group = "P 21 21 21"
}
```

We need to bring in the `master_params` again because all the meta-information was lost in the `extract()` step that produced `params`. Again, a type-specific converter is used to produce a string for each Python object. We started out with `space_group=19` but get back `space_group = "P 21 21 21"` because we chose to make the converter work that way.

2.8 Extending Phil

The astute reader may have noticed that we used both `libtbx.phil` and `iotbx.phil`. Why does Phil appear to have two homes?

The best way to think about Phil is to say "Phil is `libtbx.phil`." The basic Phil objects storing the parsing results (`phil.definition` and `phil.scope`), the tokenizer, parser and the command line support are implemented in the `libtbx.phil` module. `iotbx.phil` *extends* Phil by adding two new types, `unit_cell` and `space_group`. The converters for these types can be found in `$IOTBX_DIST/iotbx/phil.py`. For example, this is the code for the unit cell converters:

```
class unit_cell_converters:

    def __str__(self): return "unit_cell"

    def from_words(self, words, master):
        s = libtbx.phil.str_from_words(words=words)
        if (s is None): return None
        return uctbx.unit_cell(s)
```

```
def as_words(self, python_object, master):
    if (python_object is None):
        return [tokenizer.word(value="None")]
    return [tokenizer.word(value="%.10g" % v)
            for v in python_object.parameters()]
```

Arbitrary new types can be added to Phil by defining similar converters. If desired, the built-in converters for the basic types (`int`, `float`, `str`, etc.) defined in `libtbx.phil` can even be replaced. All converters have to have `__str__()`, `from_words()` and `as_words()` methods. More complex converters may optionally have a non-trivial `__init__()` method (an example is the `choice_converters` class in `$LIBTBX_DIST/libtbx/phil/__init__.py`).

The `iotbx.phil.parse()` function used in the examples above is a very small function which adds the `unit_cell` and `space_group` converters to Phil's default converter registry and then calls the main `libtbx.phil.parse()` function to do the actual work. Following the example of `iotbx.phil` it should be straightforward to add other domain-specific types to the Phil system.

2.9 Variable substitution

Phil supports shell-like variable substitution using `$var` and `${var}` syntax. A few examples say more than many words:

```
import libtbx.phil

params = libtbx.phil.parse("""
    root_name = peak
    file_name = $root_name.mtz
    full_path = $HOME/$file_name
    related_file_name = ${root_name}_data.mtz
    message = "Reading $file_name"
    as_is = ' $file_name '
    """)
params.fetch(source=params).show()
```

Output:

```
root_name = peak
file_name = "peak.mtz"
full_path = "/net/cci/rwgk/peak.mtz"
related_file_name = "peak_data.mtz"
message = "Reading peak.mtz"
as_is = ' $file_name '
```

Note that the variable substitution does not happen during parsing. The output of `params.show()` is identical to the input. In the example above, variables are substituted by the `fetch()` method that we introduced earlier to merge user files given a master file.

2.10 Phil odds and ends

Phil also supports merging of files at the parsing level. The syntax is simply `include file_name`. `include` directives may appear inside scopes to enable hierarchical building of master files without the need to copy-and-paste large fragments explicitly. Duplication appears only in automatically generated user files. I.e. the programmer is well served because a system of master files can be kept free of large-scale redundancies that are difficult to maintain. At the same time the end user is well served because the indirections are resolved automatically and all parameters are presented in one uniform view.

Variable substitution and include directives smell almost like programming. However, there is a line that Phil is never meant to cross: flow control is not a part of the syntax. It is hard to imagine that a fully featured programming language could be syntactically simpler than Python. For example, there are good reasons why Python string literals have to be quoted. Otherwise Python scripts would be full of \$ signs because some method is needed to distinguish strings from variable names. On the other hand, having to quote space group symbols in parameter files is a nuisance. In the future we may extend Phil as an interchange format for data other than parameters but for our programming needs we feel extremely well served by Python.

3 Refinement tools

3.1 mmtbx.refinement.f_model.manager

The goal of crystallographic structure refinement is to optimize a set of model parameters such that the model predictions best fit the experimental observations. In our terminology, *model* goes beyond atomic coordinates, displacement parameters and occupancies. A complete macromolecular model generally also includes other contributions such as scale factors, bulk-solvent correction and anisotropy correction. Furthermore, all modern refinement programs include facilities for cross-validation (e.g. for the calculation of the *R-free*).

The `phenix.refine` command mentioned earlier is based on the *mtbx* (*Macromolecular toolbox*) module of the *cctbx*. The mathematical foundation of the *mtbx* model parameterization is described in Afonine *et al.* (2005). It is summarized in this formula:

$$\mathbf{F}^{\text{model}} = k \exp\left(-\mathbf{h}^t \mathbf{U}_{\text{aniso}} \mathbf{h}\right) \left(\mathbf{F}^{\text{calc}} + k_{\text{sol}} \exp\left(-\frac{B_{\text{sol}} s^2}{4}\right) \mathbf{F}^{\text{mask}} \right)$$

where k is the overall scale factor (Sheriff & Hendrickson, 1987), \mathbf{F}^{calc} are structure factors calculated from the atomic model, k_{sol} and B_{sol} are bulk-solvent parameters (Jiang & Brünger, 1994), \mathbf{F}^{mask} are structure factors calculated from a molecular mask, \mathbf{h} is a column vector with the Miller indices of a reflection, \mathbf{h}^t is its transposed vector, and $\mathbf{U}_{\text{aniso}}$ is the overall anisotropic scale matrix (6 components).

During refinement, $\mathbf{F}^{\text{model}}$ is usually calculated many times in different contexts, many parameters are updated at different schedules, and various statistics are printed repeatedly to report the refinement progress. Moreover, some refinement strategies require complete sets of intermediate parameters to be stored for later reference. To meet these needs in a general and reusable way, all model parameters for the crystallographic contribution to the refinement target are grouped by the `mtbx.refinement.f_model.manager` class. In the following we develop a self-contained Python script to highlight major features of this class. Since we need data to work with, but also want the example to be self-contained, we start by generating a random structure:

```

from cctbx.development import random_structure
from cctbx import sgtbx

space_group_info = sgtbx.space_group_info(
    symbol="P212121")
n_sites = 500
structure = random_structure.xray_structure(
    space_group_info = space_group_info,
    elements          = ["N"]*(n_sites),
    volume_per_atom   = 50,
    anisotropic_flag   = False,
    random_u_iso       = True)

```

We use this `structure` to compute ideal observations `f_obs`:

```

d_min = 2.0
f_obs = abs(structure.structure_factors(
    d_min          = d_min,
    anomalous_flag = False).f_calc())

```

Next we introduce two types of errors: missing atoms and coordinate errors with a certain `max_shift`:

```

from cctbx import xray

fraction_missing = 0.1
max_shift = 0.2
n_keep = int(round(structure.scatterers().size()
    * (1-fraction_missing)))
partial_structure = xray.structure(
    special_position_settings=structure)
partial_structure.add_scatterers(
    structure.scatterers()[0:n_keep])
partial_structure.replace_scatterers(
    partial_structure.random_shift_sites(
        max_shift_cart=max_shift).scatterers())

```

As before we compute structure factors, this time for the `partial_structure`:

```

f_calc = partial_structure.structure_factors(
    d_min          = d_min,
    anomalous_flag = False).f_calc()

```

For our demonstration we need an array of R-free flags (also known as a test set). We could generate the R-free flags in one line, but we break the code up for clarity:

```

from cctbx.array_family import flex

n_reflections = f_calc.data().size()
partitioning = flex.random_permutation(size=n_reflections) % 10

```

At this point `partitioning` is an integer array with randomly assigned but uniformly distributed values from 0 to 9. Insert `print list(partitioning)` to display the array. The next line turns this integer array into a boolean array. At the same time we build a `cctbx.miller.array` (Newsletter No. 1) with the same indexing set as `f_obs` but with the boolean array as the data:

```
r_free_flags = f_obs.array(data=(partitioning == 0))
```

Finally we have all the pieces we need to initialize the main object of this demonstration:

```
import mmtbx.refinement.f_model

f_model_manager = mmtbx.refinement.f_model.manager(
    f_calc = f_calc,
    f_obs = f_obs,
    r_free_flags = r_free_flags)
f_model_manager.show()
```

The output of the `show()` method is:

```
f_calc      = <cctbx.miller.array object at 0xb5e9ff6c>
f_obs       = <cctbx.miller.array object at 0xb60e170c>
f_mask      = <cctbx.miller.array object at 0xb5eccb4c>
r_free_flags = <cctbx.miller.array object at 0xb5e9ff0c>
u_aniso     = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
k_sol       = 0.0
b_sol       = 0.0
scale_work  = 1.0
scale_test  = 1.0
alpha       = None
beta        = None
sf_algorithm = None
target_name = None
target_functors = None
```

Our `f_model_manager` maintains references to the input arrays (`f_calc`, `f_obs`, `r_free_flags`). We also see a new `f_mask` array used for bulk-solvent correction and all the parameters introduced above. Some parameters are not defined (`None`), but these are not needed in this example. As is, the `f_model_manager` is already able to answer certain questions, for example, what are the current values of R-work and R-free:

```
print f_model_manager.r_work()
print f_model_manager.r_free()
```

Output:

```
0.275794965768
0.28068823468
```

(The output may vary since we are working with a random structure.) More detailed information is just waiting for us:

```
f_model_manager.r_factors_in_resolution_bins(
    reflections_per_bin = 100,
    max_number_of_bins  = 10)
```

Output:

Bin number	Resolution		No. Refl.		R-factors	
	range		work	test	work	test
1:	20.6581	- 3.8191	988	111	0.2778	0.2555
2:	3.8191	- 3.0344	924	114	0.2596	0.2610
3:	3.0344	- 2.6517	914	107	0.2649	0.2750
4:	2.6517	- 2.4097	899	104	0.2713	0.2887
5:	2.4097	- 2.2372	906	95	0.2856	0.2998
6:	2.2372	- 2.1054	927	80	0.2992	0.2971
7:	2.1054	- 2.0001	884	106	0.2846	0.3438

If model parameters are updated the `f_model_manager` automatically recomputes all dependent values:

```
f_model_manager.update(  
    k_sol = 1.2,  
    b_sol = 30.0)
```

This centralized, concise facility is extremely helpful in developing new refinement strategies.

At any stage, F^{model} according to the formula above, or just the bulk-solvent correction can easily be extracted:

```
f_model = f_model_manager.f_model()  
f_bulk = f_model_manager.f_bulk()
```

Detailed and uniform statistics can easily be displayed in various contexts. For example:

```
f_model_manager.show_fom_phase_error_alpha_beta_in_bins(  
    reflections_per_bin = 100,  
    max_number_of_bins = 10)
```

Output:

```
|-----|  
|R-free likelihood based estimates for figures of merit, absolute phase error,|  
|and distribution parameters alpha and beta (Acta Cryst. (1995). A51, 880-887)|  
|  
| Bin      Resolution      No. Refl.   FOM   phase err.   Alpha      Beta |  
|number    range          work    test    <|p-p c|> |  
| 1: 20.6581 - 3.8191    988     111   0.8414   20.7174    0.9663    5782.0632|  
| 2: 3.8191 - 3.0344    924     114   0.8190   23.9808    0.9663    5782.0632|  
| 3: 3.0344 - 2.6517    914     107   0.8208   24.0338    0.9372    4108.1081|  
| 4: 2.6517 - 2.4097    899     104   0.8058   25.6936    0.9226    3272.3595|  
| 5: 2.4097 - 2.2372    906      95   0.7733   28.8366    0.9251    3055.6162|  
| 6: 2.2372 - 2.1054    927      80   0.7806   28.1986    0.9304    2583.5975|  
| 7: 2.1054 - 2.0001    884     106   0.7484   31.1348    0.9304    2583.5975|  
|-----|
```

After refinement is is often very helpful to inspect electron density maps. Since the `f_model_manager` controls all essential data for the calculation of maps, it is most natural to add a map generation method. For example:

```
fft_map = f_model_manager.electron_density_map(
    map_type = "2m*Fobs - alpha*Fmodel")
```

Or:

```
fft_map = f_model_manager.electron_density_map(
    map_type = "k*Fobs - n*Fmodel",
    k         = 2,
    n         = 1)
```

To end this demonstration, we bring in the *iotbx* utilities for writing maps in XPLOR format:

```
import iotbx.xplor.map

fft_map.as_xplor_map(
    file_name="2fo-fm.xplor",
    title_lines=["2*Fobs - Fmodel"],
    gridding_first=(0,0,0),
    gridding_last=fft_map.n_real())
```

Happy viewing! -- Well, admittedly it is not very interesting to view maps of random structures, but it works just the same given real data and real models.

The complete script can be found in the *cctbx* installation:

```
$MMTBX_DIST/mmtbx/examples/f_model_manager.py
```

3.2 Bulk-solvent correction and anisotropic scaling

In the previous issue of the Newsletter (No. 3) we briefly described a protocol for the determination of flat bulk-solvent model parameters and anisotropic scaling parameters. In the current version of the *cctbx* we have generalized this protocol significantly. The main features currently available are:

1. In addition to the least-squares target function presented before, a maximum-likelihood crystallographic target function can be used for the determination of the bulk-solvent and scale parameters. This enables a uniform overall strategy for maximum-likelihood model refinement since all parameters (bulk solvent, scale and atomic) can be refined against the same target function.
2. Three options for defining the bulk-solvent parameters (k_{sol} , B_{sol}) and the anisotropic scale matrix $\mathbf{U}_{\text{aniso}}$:
 - a. Manual assignment. This is potentially useful at the beginning of structure refinement when the model has many errors.
 - b. Minimization of a crystallographic target function using the *LBFGS* minimizer. This is a quick and precise way of determining k_{sol} , B_{sol} and $\mathbf{U}_{\text{aniso}}$ if a model of reasonable quality is available and the experimental data extend to sufficiently low resolution. However, this algorithm fails to produce physically reasonable parameters in some situations. This experience was the motivation for implementing the more sophisticated protocol outlined below.
 - c. Combined *LBFGS* minimization and grid search algorithm (Afonine *et al*, 2005). This is the most robust procedure for the determination of k_{sol} , B_{sol} and $\mathbf{U}_{\text{aniso}}$. However, it is also the most time-consuming option.

The bulk-solvent and scaling algorithms are implemented in the `mmtbx.bulk_solvent` module.

3.3 Simulated annealing refinement

Simulated annealing is a time-tested tool for escaping local minima in crystallographic refinement (Brünger *et al.*, 1987). Recently we have implemented a simulated annealing algorithm for restrained molecular dynamics in the *cctbx*. This enables us to take full advantages of combined simulated annealing and maximum-likelihood model refinement (Adams *et al.*, 1997; Brunger & Adams, 2002).

The simulated annealing algorithms are implemented in the `mmtbx.cartesian_dynamics` module.

3.4 Building of hydrogen atoms

Fourier syntheses at subatomic resolution ($d_{\min} < 1.0 \text{ \AA}$) usually reveal the presence of hydrogen atoms. At lower resolutions this information is lost. Therefore a general refinement program has to provide different strategies depending on the resolution of the data. If ultra-high resolution data are available, hydrogens can be explicitly included in the refinement, for example using the riding hydrogen model (Sheldrick, 1995). At lower resolutions the inclusion of hydrogens in the refinement target for the X-ray data is likely to lead to overfitting. However in this case the hydrogens should still be considered in the definition of the geometry restraints, and this has been shown to improve atomic models even in the absence of atomic resolution data (Richardson *et al.* 2003). In addition, refinement against neutron diffraction data requires appropriate modeling of hydrogen atoms.

As a first step towards covering these cases we have implemented a hydrogen building procedure for the standard amino acid residues. In most cases the hydrogen positions are geometrically well defined. However, there are some cases where the positions are not unambiguously determined, such as -CH₃, -OH in a tyrosine residue. To account for this, our procedure consists of two steps. In the first step we place all expected hydrogen atoms in appropriate positions. If ambiguities exist, we place the affected hydrogens arbitrarily in a one of the allowed positions. In the second step we perform model regularization by refinement against geometry restraints (see Newsletter No. 4). Optionally, this can be combined with Cartesian dynamics to escape from local minima.

The hydrogen building algorithms are implemented in the `mmtbx.hydrogens` module.

3.5 Maximum-likelihood tools

Previously we had implemented an amplitude-based maximum-likelihood target function (Lunin *et al.*, 2002), its quadratic approximation (Lunin & Urzhumtsev, 1999), and a procedure for estimating the distribution parameters (alpha, beta) according to Lunin & Skovoroda (1995). Recently we have extended the set of maximum-likelihood tools by these methods:

R-free likelihood-based estimation of model phase errors and figures of merit

This procedure is based on the algorithm described by Lunin & Skovoroda (1995). The mean phase errors and figures of merit are determined in narrow resolution bins using test reflections only. The procedure provides relatively precise and unbiased values for these parameters. The algorithms are available via methods of the `mmtbx.refinement.f_model.manager` class introduced in section 3.1, e.g.:

```
figures_of_merit = f_model_manager.figures_of_merit()
phase_errors = f_model_manager.phase_errors()
```

Coefficients for Fourier Syntheses

It was straightforward to implement the calculation of “best” coefficients for Fourier syntheses, $[2m_s F_s^{\text{obs}} - \alpha_s F_s^{\text{model}}] \exp(i\varphi_s^{\text{calc}})$, where m_s are figures of merit and $\alpha_s = \langle \cos(\mathbf{s}, \Delta \mathbf{r}) \rangle$ (Urzhumtsev et al., 1996; Read, 1986 uses the notation D_s). The `f_model_manager.electron_density_map()` method demonstrated in section 3.1 provides an interface to these algorithms.

Use of Experimental Phase Information

We are actively working on fast C++ code for a maximum-likelihood target which includes experimental phase information (MLHL target; Pannu *et al*, 1998). This code is in the *cctbx* bundles already but not yet fully tested.

4 `iotbx.reflection_statistics`

Recently we have enhanced the `iotbx.reflection_statistics` command significantly. The initial version (written in April 2004) can be used to compute data completeness, anomalous signals, correlations between intensities and correlations between anomalous signals of pairs of reflection arrays. All these statistics are computed both in resolution shells and as overall quantities. The latest version (written in December 2004) adds these new features:

- Automatic determination of the space group of the metric (i.e. the lattice symmetry; see also Newsletter No. 3).
- Automatic derivation of a non-redundant set of possible twin laws from first principles (Flack, 1987).
- Automatic derivation of a non-redundant set of possible reindexing matrices for comparing two datasets. The matrices are derived from first principles (see below).
- Computation of a sorted list of peaks in the native Patterson synthesis to facilitate the detection of translational non-crystallographic symmetry (NCS).
- Tests for perfect merohedral twinning using both the second moments of amplitudes (also known as Wilson ratios) and intensities (Yeates, 1997).

With the old version of the `iotbx.reflection_statistics` command correlations between pairs of reflection arrays are computed only if the unit cell parameters and the space group symmetries are identical. The new version is designed to overcome this limitation in the most general way. Internally, all arrays are transformed to a primitive setting. The change-of-basis matrices are determined with a cell reduction algorithm (see Newsletter No. 3). Each array in the primitive setting is expanded to P1. I.e. the symmetry matrices are applied to generate all equivalent Miller indices. Given a pair of reflection arrays preprocessed in this way, a newly developed algorithm performs an exhaustive search for the change-of-basis matrix that leads to the best superposition of the reduced unit cells. This algorithm employs the new `similarity_transformations()` and `bases_mean_square_difference()` methods of the `cctbx.uctbx.unit_cell` class. Associated with each unit cell is the space group of the metric as determined with the lattice symmetry algorithm outlined in the Newsletter No. 3. If the tolerances used in the computation of the cell superposition are reasonable, the metric symmetries are identical, or one is a subgroup of the other. We continue with the highest metric space group. Each symmetry operation of this space group is a possible reindexing matrix. Conceptually, we compute the correlations between two arrays for each reindexing matrix and produce a sorted list of the results. However, if any of the space groups of the two input arrays are different from P1, this leads to a redundant list. To remove these redundancies, we employ double coset decomposition (see below). To minimize the runtime, redundant correlations are never computed.

The algorithmic complexities are in stark contrast to the simple end-user interface. The universal reflection file reader described in Newsletter No. 3 is used to automatically detect and process all common file formats. A possible command for comparing reflection data is:

```
iotbx.reflection_statistics *.sca *.mtz
```

For a large number of arrays this may take a couple of minutes, but the comprehensive analyses do not require any user intervention. The potentially large output contains tags for quick searching. A guide is printed at the beginning of the output. For example:

```
Array indices (for quick searching):
 1: hg1.0_scale_anomalous.sca:i_obs,sigma
 2: hginfl_3.5_ano.sca:i_obs,sigma
 3: hgpeak_3.5_ano.sca:i_obs,sigma
 4: pb1.0_4.0_ano.sca:i_obs,sigma
 5: pbpeak_3.5_scale_anomalous.sca:i_obs,sigma
 6: pt_4.0_ano.sca:i_obs,sigma
 7: scale.sca:i_obs,sigma
 8: sm_scale_anomalous.sca:i_obs,sigma
 9: tmpb.sca:i_obs,sigma

Useful search patterns are:
  Summary i
  CC Obs i j
  CC Ano i j
  i and j are the indices shown above.
```

If we search for `CC Obs 7 1` we find:

```
CC Obs 7 1  0.956 h,-k,-l
Correlation of:
  scale.sca:i_obs,sigma
  hg1.0_scale_anomalous.sca:i_obs,sigma
Overall correlation with reindexing:  0.956 h,-k,-l
unused:      - 43.6948 [  4/20 ]  1.000
bin  1: 43.6948 -  8.6072 [2856/2950]  0.954
bin  2:  8.6072 -  6.8402 [2916/2924]  0.962
bin  3:  6.8402 -  5.9780 [3028/3036]  0.957
bin  4:  5.9780 -  5.4326 [2936/2948]  0.960
bin  5:  5.4326 -  5.0438 [2940/2960]  0.964
bin  6:  5.0438 -  4.7468 [2792/2818]  0.959
bin  7:  4.7468 -  4.5093 [3104/3124]  0.954
bin  8:  4.5093 -  4.3132 [2824/2842]  0.949
bin  9:  4.3132 -  4.1473 [2904/2930]  0.946
bin 10:  4.1473 -  4.0043 [2964/2990]  0.937
unused:  4.0043 -           [ 24/72 ]  0.904

CC Obs 7 1  0.364 h,k,l
Correlation of:
  scale.sca:i_obs,sigma
  hg1.0_scale_anomalous.sca:i_obs,sigma
Overall correlation:  0.364
```

In this example the highest correlation (0.956) between the two arrays is found with the reindexing matrix `h,-k,-l`. In contrast, the correlation between the arrays as indexed originally is only 0.364.

The `iotbx.reflection_statistics` command is implemented in the file `$IOTBX_DIST/iotbx/command_line/reflection_statistics.py`.

4.1 Double coset decomposition

A useful summary of the theory of double cosets can be found in *An introduction to group theory* by Tony Gaglione, which is available online:

<http://web.usna.navy.mil/~wdj/tonybook/gpthry/node44.html>

Double coset decomposition is concerned with a group `g` and two subgroups `h1` and `h2`. The group `g` is partitioned into non-overlapping sets of symmetry operations equivalent under `h1` and `h2`. In the context of the algorithm outlined above, `g` is the highest space group of the metric. `h1` and `h2` are the space groups of the arrays to be compared. Each double coset represents a reindexing choice unique under `h1` and `h2`. I.e. any matrix selected from a given double coset will lead to identical correlation coefficients.

If we do not care which matrix is selected from a given double coset, we arrive at a surprisingly simple algorithm. The following is the relevant fragment from the file `$CCTBX_DIST/cctbx/sgtbx/cosets.py`:

```
def double_unique(g, h1, h2):
    result = []
    done = {}
    for a in g:
        if (str(a) in done): continue
        result.append(a)
        for hi in h1:
            for hj in h2:
                b = hi.multiply(a).multiply(hj)
                done[str(b)] = None
    return result
```

`g`, `h1` and `h2` are instances of `cctbx.sgtbx.space_group`. The algorithm follows directly from the definition of cosets as found at the web page referenced above:

For `a`, `b` element of `g`, we define `a ~ b` if and only if `h1 a h2 = b`.

`h1 a h2 = b` corresponds to `b = hi.multiply(a).multiply(hj)` in the Python code.

`result` is a Python list of representative matrices, one from each coset. Which matrices are returned depends on the order of the matrices in `g`, `h1` and `h2`. This may not always yield the "nicest" choice. However, any investment in a more sophisticated selection has little or no practical value. Typically the transformed indices are mapped into a canonical asymmetric unit (e.g. using the `map_to_asu()` method of `cctbx.miller.array`). After this manipulation the indexing set will be the same no matter which matrix from a given double coset is selected.

5 iotbx.mtz

CCP4 MTZ files are binary files containing merged or unmerged reflection data and optionally information about raw data ("batches"). For a couple of years already the `cctbx` has included C++ and Python interfaces to the CCP4 C MTZ library in the `iotbx.mtz` module. However, while the support for reading MTZ files was quite complete, creating and writing MTZ files was only partially supported. To

resolve this problem and to unify the interfaces for reading and writing, the `iotbx.mtz` module was heavily restructured. We have also added complete C++ and Python interfaces for the manipulation of MTZ batches. The `iotbx.mtz` module extends the functionality of the CCP4 C MTZ library by automatically grouping related MTZ columns into one object, `cctbx.miller.array` instances as introduced in Newsletter No. 1.

Combined with the universal reflection file reader, it is quite easy to quickly write a script for converting any of the formats processed by the reflection file reader to the MTZ format. First let's get some data to work with:

```
from iotbx import reflection_file_reader
import os

reflection_file = reflection_file_reader.any_reflection_file(
    file_name=os.path.expandvars(
        "$CNS_SOLVE/doc/html/tutorial/data/pen/scale.hkl"))
```

We are reading a CNS reflection file in the CNS tutorial. (To run this example CNS has to be installed including the tutorial.) Since the crystal symmetry is not defined in CNS reflection files, we supply this information manually:

```
from cctbx import crystal

crystal_symmetry = crystal.symmetry(
    unit_cell=(97.37, 46.64, 65.47, 90, 115.4, 90),
    space_group_symbol="C2")
```

We convert the reflection file to a list of `cctbx.miller.array` instances:

```
miller_arrays = reflection_file.as_miller_arrays(
    crystal_symmetry=crystal_symmetry)
```

Now we loop over the Miller arrays to convert them to MTZ data columns:

```
mtz_dataset = None
for miller_array in miller_arrays:
    if (mtz_dataset is None):
        mtz_dataset = miller_array.as_mtz_dataset(
            column_root_label=miller_array.info().labels[0])
    else:
        mtz_dataset.add_miller_array(
            miller_array=miller_array,
            column_root_label=miller_array.info().labels[0])
```

Let's see what we got:

```
mtz_object = mtz_dataset.mtz_object()
mtz_object.show_summary()
```

The output ends with:

```
Column number, label, number of valid values, type:
 1 H          6735/6735=100.00% H: index h,k,l
 2 K          6735/6735=100.00% H: index h,k,l
 3 L          6735/6735=100.00% H: index h,k,l
 4 F_PHGA     6735/6735=100.00% F: amplitude
 5 SIGF_PHGA  6735/6735=100.00% Q: standard deviation
 6 F_KUOF     6735/6735=100.00% F: amplitude
 7 SIGF_KUOF  6735/6735=100.00% Q: standard deviation
 8 F_NAT      6735/6735=100.00% F: amplitude
 9 SIGF_NAT   6735/6735=100.00% Q: standard deviation
```

Finally we write the MTZ file to disk:

```
mtz_object.write("pen_data.mtz")
```

Note that the `iotbx.mtz.dump pen_data.mtz` command is available to produce the same output as the `mtz_object.show()` statement in the example.

6 Integration of PyCifRW

PyCifRW is a library for reading and writing CIF (Crystallographic Information Format) files using Python. PyCifRW was developed by James Hester at the Australian National Beamline Facility (ANBF). Documentation can be found online:

<http://www.ansto.gov.au/natfac/ANBF/CIF/>

Recently, the PyCifRW license was changed to allow redistribution. We are very excited about this development because it allows us to include PyCifRW in the *cctbx* bundles. However, like the CCP4 I/O library and Clipper (see Newsletter No. 4), PyCifRW is not in the *cctbx* CVS tree on SourceForge. James Hester continues to develop PyCifRW in his own environment and we will update the *cctbx* bundles with the latest releases. Currently we redistribute PyCifRW version 1.19 released in November 2004.

PyCifRW in a *cctbx* installation is used in the same way as described in the PyCifRW documentation. Let's try it out. We develop a self-contained Python script by starting with embedded CIF syntax:

```
file("quartz.cif", "w").write("""
data_global
 _chemical_name Quartz
 _cell_length_a 4.9965
 _cell_length_b 4.9965
 _cell_length_c 5.4570
 _cell_angle_alpha 90
 _cell_angle_beta 90
 _cell_angle_gamma 120
 _symmetry_space_group_name_H-M 'P 62 2 2'
loop_
 _atom_site_label
 _atom_site_fract_x
 _atom_site_fract_y
 _atom_site_fract_z
Si   0.50000  0.00000  0.00000
O    0.41520  0.20760  0.16667
""")
```

At this point we have created a file `quartz.cif`. Now we parse it with PyCifRW:

```
from PyCifRW.CifFile import CifFile

cif_file = CifFile("quartz.cif")
cif_global = cif_file["global"]
print cif_global["_chemical_name"]
```

Output:

```
Quartz
```

Looks like a good start! But we want more. For example, structure factors. For this we have to process the rest of the data in the CIF file. First we determine the crystal symmetry:

```
from cctbx import uctbx, sgtbx, crystal

unit_cell = uctbx.unit_cell([float(cif_global[param])
    for param in [
        "_cell_length_a", "_cell_length_b", "_cell_length_c",
        "_cell_angle_alpha", "_cell_angle_beta", "_cell_angle_gamma"]])
space_group_info = sgtbx.space_group_info(
    symbol=cif_global["_symmetry_space_group_name_H-M"])
crystal_symmetry = crystal.symmetry(
    unit_cell=unit_cell,
    space_group_info=space_group_info)
crystal_symmetry.show_summary()
```

Output:

```
Unit cell: (4.9965, 4.9965, 5.457, 90, 90, 120)
Space group: P 62 2 2 (No. 180)
```

Now we turn our attention to the list of coordinates and create a new `cctbx.xray.structure` instance:

```
from cctbx import xray

structure = xray.structure(crystal_symmetry=crystal_symmetry)
for label,x,y,z in zip(cif_global["_atom_site_label"],
    cif_global["_atom_site_fract_x"],
    cif_global["_atom_site_fract_y"],
    cif_global["_atom_site_fract_z"]):
    scatterer = xray.scatterer(
        label=label,
        site=[float(s) for s in [x,y,z]])
    structure.add_scatterer(scatterer)
structure.show_summary().show_scatterers()
```


Output:

```
Number of scatterers: 2
At special positions: 2
Unit cell: (4.9965, 4.9965, 5.457, 90, 90, 120)
Space group: P 62 2 2 (No. 180)
Label, Scattering, Multiplicity, Coordinates, Occupancy, Uiso
Si   Si       3 ( 0.5000  0.0000  0.0000) 1.00 0.0000
O    O       6 ( 0.4152  0.2076  0.1667) 1.00 0.0000
```

Just one more hoop and we have the structure factors:

```
f_calc = structure.structure_factors(d_min=2).f_calc()
abs(f_calc).show_summary().show_array()
```

Output:

```
Miller array info: None
Observation type: None
Type of data: double, size=7
Type of sigmas: None
Number of Miller indices: 7
Anomalous flag: False
Unit cell: (4.9965, 4.9965, 5.457, 90, 90, 120)
Space group: P 62 2 2 (No. 180)
(1, 0, 0) 15.708493924
(1, 0, 1) 36.2626337008
(1, 0, 2) 7.77312576362
(1, 1, 0) 14.9039425672
(1, 1, 1) 0.975009858138
(2, 0, 0) 15.8407980479
(2, 0, 1) 13.6738859288
```

Note that this is almost what we had in Newsletter No. 1. The main difference is that we start from a CIF file rather than the plain *cctbx* interfaces.

The complete script can be found in the *cctbx* installation:

```
$PYCIFRW_DIST/example_quartz.py
```

7 Acknowledgments

We like to thank James Hester for writing PyCifRW and for his hard work concerning the PyCifRW license. We gratefully acknowledge the financial support of NIH/NIGMS. Our work was supported in part by the US Department of Energy under Contract No. DE-AC03-76SF00098.

8 References

- Adams, P. D., Pannu, N. S., Read, R. J. & Brünger, A. T. (1997). *Proc. Natl. Acad. Sci.* **94**, 5018-5023.
- Afonine, P.V., Grosse-Kunstleve, R.W. & Adams, P. D. (2005). Submitted.
- Brünger, A. T & Adams, P. D. (2002). *Acc. Chem. Res.* **35**, 404-412.
- Brünger, A. T., Kuriyan, J., Karplus, M. (1987). *Science*. **235**, 458- 460.
- Flack, H.D. (1987). *Acta Cryst.* **A43**, 564-568.
- Grosse-Kunstleve, R.W., Adams, P.D. (2003). *Newsletter of the IUCr Commission on Crystallographic Computing*, 1, 28-38. <http://www.iucr.org/iucr-top/comm/ccom/newsletters/2003jan/>
- Grosse-Kunstleve, R.W., Sauter, N.K., Adams, P.D. (2004). *Newsletter of the IUCr Commission on Crystallographic Computing*, 3, 22-31. <http://www.iucr.org/iucr-top/comm/ccom/newsletters/2004jan/>
- Grosse-Kunstleve, R.W., Afonine, P.V., B., Adams, P.D. (2004). *Newsletter of the IUCr Commission on Crystallographic Computing*, 4, 19-36. <http://www.iucr.org/iucr-top/comm/ccom/newsletters/2004aug/>
- Jiang, J.-S. & Brünger, A. T. (1994). *J. Mol. Biol.* **243**, 100-115.
- Lunin, V.Y. & Skovoroda, T.P. (1995). *Acta Cryst.* **A51**, 880-887.
- Lunin, V.Y. & Urzhumtsev, A. (1999). *CCP4 Newsletter on Protein Crystallography*, **37**, 14-28.
- Lunin, V.Y., Afonine, P.V. & Urzhumtsev, A. (2002). *Acta Cryst.*, **A58**, 270-282.
- Pannu, N. S., Murshudov, G. N., Dodson, E. J. & Read, R. J. (1998). *Acta Cryst.* **D54**, 1285-1294.
- Read, R.J. (1986). *Acta Cryst.* **A42**, 140-149.
- Richardson, J.S., Arendall, W.B. III, and Richardson, D.C. (2003). *Methods Enzymol.* **374**, 385-412.
- Sheldrick, G.M. (1985). SHELXS86. Program for the Solution of Crystal Structures. Univ. of Göttingen, Germany.
- Sheriff, S. & Hendrickson, W. A. (1987). *Acta Cryst.* **A43**, 118-121.
- Urzhumtsev, A.G., Skovoroda, T.P., Lunin, V.Y. (1996). *J. Appl. Cryst.* **29**, 741-744.
- Yeates, T.O. (1997). *Methods Enzymol.* **276**, 344-358.

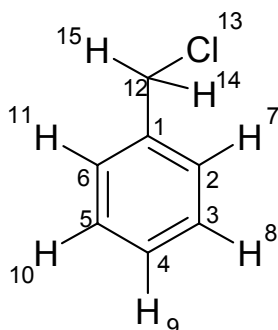
Computing the Z-matrix for global optimisation

Kenneth Shankland,

ISIS Facility, Rutherford Appleton Lab., Oxon OX11 0QX, U.K. - Email : K.Shankland@rl.ac.uk ;
WWW: <http://www.isis.rl.ac.uk/dataanalysis/people/ken/KEN.HTM>

Introduction

In a previous article [1] the construction of internal coordinate molecular models, suitable for use in global optimisation schemes against powder diffraction data, was discussed. In particular, the 'Z-matrix' format was outlined in some detail and the importance of defining flexible torsion angles (i.e. torsion angles around which free rotation can occur) as a series of proper and improper torsions was emphasised. Many computer programs can generate a Z-matrix from a given input molecular model but all those examined by this author (admittedly a few years ago!) did not produce Z-matrices with torsion angles defined in this fashion. By way of example, consider the following simple structure:



The CH₂Cl group can rotate around bond 1-12. If we assume for a moment that the chlorine atom lies in the plane of the ring, then one Z-matrix that would describe this molecule is;

C1	0.0000000	0	0.0000000	0	0.0000000	0	0	0	0
C2	1.4000000	0	0.0000000	0	0.0000000	0	1	0	0
C3	1.4000000	0	120.0000000	0	0.0000000	0	2	1	0
C4	1.4000000	0	120.0000000	0	0.0000000	0	3	2	1
C5	1.4000000	0	120.0000000	0	0.0000000	0	4	3	2
C6	1.4000000	0	120.0000000	0	0.0000000	0	5	4	3
H7	1.0000000	0	120.0000000	0	180.0000000	0	2	3	4
H8	1.0000000	0	120.0000000	0	180.0000000	0	3	4	5
H9	1.0000000	0	120.0000000	0	180.0000000	0	4	5	6
H10	1.0000000	0	120.0000000	0	180.0000000	0	5	6	1
H11	1.0000000	0	120.0000000	0	180.0000000	0	6	1	2
C12	1.4000000	0	120.0000000	0	180.0000000	0	1	2	3
C113	1.7000000	0	109.5000000	0	0.0000000	0	12	1	2
H14	1.0000000	0	109.5000000	0	120.0000000	0	12	1	2
H15	1.0000000	0	109.5000000	0	240.0000000	0	12	1	2

Whilst this is satisfactory for this single conformation of the molecule (and therefore a good starting point for say, a single-point energy calculation) it is clear that if we want the ability to generate any permissible conformation about bond 1-12 (as we do, for global optimisation), we need to change the last three lines to:

C113	1.7000000	0	109.5000000	0	0.0000000	1	12	1	2
H14	1.0000000	0	109.5000000	0	120.0000000	0	12	1	13
H15	1.0000000	0	109.5000000	0	120.0000000	0	12	1	14

We therefore create a Z-matrix where any allowable value can be entered for torsion 13-12-1-2 and the attached hydrogen atoms will automatically rotate, as they are now defined *relative* to the position of the chlorine atom. However, for anything other than the simplest of molecules, it is tedious to create a Z-matrix in this form manually. Fortunately, a Cartesian XYZ file (easily produced from molecular modelling programs) with atomic connectivity has all the information needed to construct an appropriate Z-matrix automatically; the steps needed to effect this conversion are described below.

Desirable attributes for a computer program that calculates Z-matrices

- The ability to specify the starting atom for the Z-matrix
- The ability to identify flexible torsion angles. Note that all torsion angles are normally considered to be flexible except when either (a) the torsion involved is part of a ring system or (b) the torsion involved has a bond order greater than one
- The ability to identify torsions that involve only 'H' atom rotations, as rotation around these torsions will not have any significant impact upon the calculation of an X-ray powder diffraction pattern

Of these attributes, the first two are *essential* and the latter desirable.

A strategy for the calculation

Step 1

Firstly, we obtain a model of the input structure in Cartesian co-ordinates with associated atomic connectivity. For the example presented later (famotidine), CSSR format is used:

#	label	x	y	z	connectivity			
1	S1	0.00000	0.00000	0.00000	2	3	4	33
2	O2	1.43958	0.00000	0.00000	1	0	0	0
3	O3	-0.66320	1.28419	0.00000	1	0	0	0
4	N4	-0.40765	-0.88451	1.28417	1	5	0	0
5	C5	-1.67657	-1.14553	1.55244	4	6	9	0
etc. . . .								

Thus atom 1 is connected to atoms 2,3,4 and 33, Atom 2 is connected to 1 etc.....

Step 2

In terms of atom identification, we will have the original atom numbering scheme and a new numbering scheme for the atoms within the Z-matrix. Rather than attempting to maintain some correspondence between the original and the new numbering scheme, we will aim to *renumber the structure in a one-off operation*. This new numbering scheme is the one that we will then use for the Z-matrix. The renumbering algorithm is simple and will be presented later. The renumbered atom list is then ordered in ascending order, just like the original, giving, for example:

1	S11	-3.15763	0.38730	4.11686	2	3	0	0
2	C10	-1.83328	-0.83920	4.04576	1	4	5	6
3	C12	-2.46400	1.75399	3.11132	1	20	21	22
4	C9	-1.92778	-1.82462	2.87380	2	7	8	9
5	H29	-1.88968	-1.31614	4.82938	2	0	0	0

Step 3

It is clear that there is redundancy in the connectivity list that can be removed. For example, atom 1 is connected to 2 and 3, but that information is contained further down the list in the connectivities of atoms 2 and 3. Therefore, we trim the connectivity list with the following rule: *for each atom in the list, look at the connectivity list and delete any atoms that have a higher number than the current atom. For any atoms that remain in the connected list, ensure that they are placed in ascending order.* Applying this rule to the five atom list shown earlier, the list becomes

1	S11	-3.15763	0.38730	4.11686	
2	C10	-1.83328	-0.83920	4.04576	1
3	C12	-2.46400	1.75399	3.11132	1
4	C9	-1.92778	-1.82462	2.87380	2
5	H29	-1.88968	-1.31614	4.82938	2

For a compound that contains no ring systems, each atom in the list will only have a single connected atom. For a compound that has a single ring system, only one atom in the list will have two connected atoms remaining at the end of this step. This property of the list makes it easy to identify ring systems and hence identify non-rotating bonds.

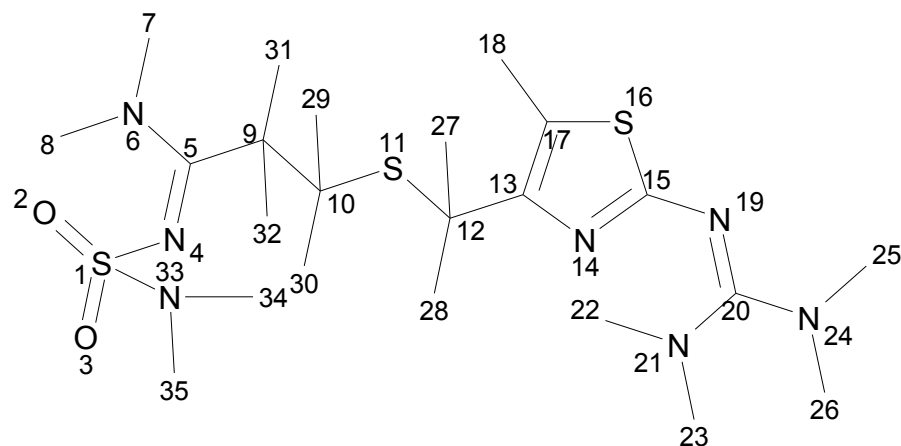
Step 4

This simplified representation makes generation of a Z-matrix straightforward, provided that the following rules are remembered

- A flexible torsion angle can only cross the central twisting bond a single time. For example, if 4-3-2-1 is a legitimate flexible torsion, 5-3-2-1 cannot be, because the 3-2 bridge has already been crossed. Instead, the position for atom 5 must be defined relative to the atom that first crossed the bridge i.e. an improper torsion 5-3-2-4.
- *If the central bond is part of a ring, it cannot be a flexible torsion.*
- *Each time we look for a 'connection' from an atom, we always use the lowest index atom that is connected, excluding the atom we are tracking from*

We'll now take a typical example (famotidine) and see how this works in practice.

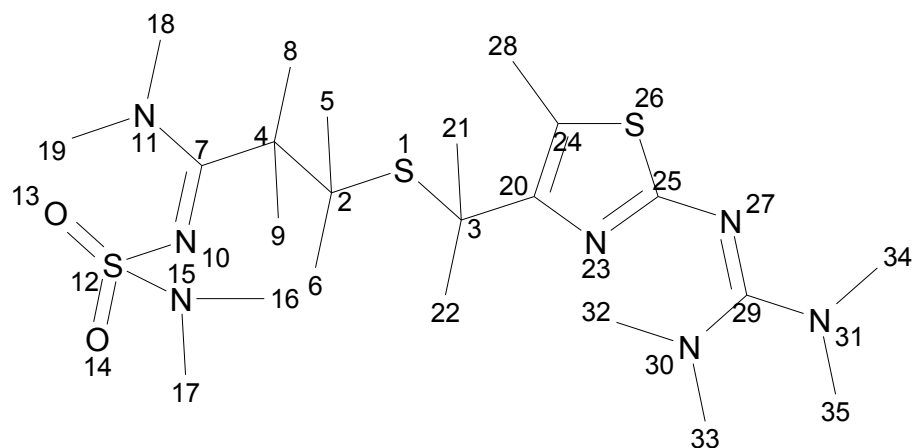
Famotidine: renumbering the structure



The numbering scheme is from a .CSSR file, which is shown below

1	S1	0.00000	0.00000	0.00000	2	3	4	33
2	O2	1.43958	0.00000	0.00000	1	0	0	0
3	O3	-0.66320	1.28419	0.00000	1	0	0	0
4	N4	-0.40765	-0.88451	1.28417	1	5	0	0
5	C5	-1.67657	-1.14553	1.55244	4	6	9	0
6	N6	-2.72661	-0.84659	0.82513	5	7	8	0
7	H7	-2.64780	-0.44281	0.13183	6	0	0	0
8	H8	-3.51664	-1.07174	1.09032	6	0	0	0
9	C9	-1.92778	-1.82462	2.87380	5	10	31	32
10	C10	-1.83328	-0.83920	4.04576	9	11	29	30
11	S11	-3.15763	0.38730	4.11686	10	12	0	0
12	C12	-2.46400	1.75399	3.11132	11	13	27	28
13	C13	-1.37355	2.49913	3.79836	12	14	17	0
14	N14	-1.72206	3.44723	4.74729	13	15	0	0
15	C15	-0.65834	4.05997	5.22388	14	16	19	0
16	S16	0.84898	3.46491	4.54237	15	17	0	0
17	C17	-0.04407	2.35168	3.57109	13	16	18	0
18	H18	0.39441	1.77470	2.93215	17	0	0	0
19	N19	-0.55717	5.05435	6.13996	15	20	0	0
20	C20	-1.65577	5.66504	6.58976	19	21	24	0
21	N21	-1.49335	6.65177	7.47900	20	22	23	0
22	H22	-0.74545	6.92387	7.68745	21	0	0	0
23	H23	-2.15564	7.03585	7.73829	21	0	0	0
24	N24	-2.90102	5.36593	6.21922	20	25	26	0
25	H25	-3.02436	4.74583	5.69903	24	0	0	0
26	H26	-3.51859	5.80870	6.51171	24	0	0	0
27	H27	-2.14046	1.40036	2.24489	12	0	0	0
28	H28	-3.18891	2.34331	2.91616	12	0	0	0
29	H29	-1.88968	-1.31614	4.82938	10	0	0	0
30	H30	-1.03619	-0.39013	4.03420	10	0	0	0
31	H31	-1.26103	-2.49122	2.97977	9	0	0	0
32	H32	-2.82899	-2.21138	2.84944	9	0	0	0
33	N33	-0.58734	-0.79346	-1.29434	1	34	35	0
34	H34	-0.72390	-0.27427	-1.94887	33	0	0	0
35	H35	-0.11417	-1.48289	-1.51453	33	0	0	0

This molecule is numbered starting from one end. Ideally, we would like the Z-matrix to start from S11 which lies at the middle of the molecule - this ensures that dependencies amongst the torsion angles are evenly distributed, which confers performance benefits during the global optimisation stage [2]. Thus we will renumber the structure based upon the connectivity, starting from atom S11. The algorithm is straightforward: pick the start atom and call it atom 1 and make a note that this has been renumbered and *traced*. Look at the n atoms connected to it and number them 2 through to n . Make a note that these atoms have been renumbered. Now loop over the connected atoms and look at the atoms connected to those atoms i.e. begin *tracing* the first connected atom. Clearly, this is a recursive process which will continue until there are no more atoms left to trace that are ‘descended’ from this first connected atom. The initial loop over the atoms connected to atom 1 will then move on to the second connected atom and so on until all atoms have been renumbered and all atoms have been marked as having been *traced*. A trivial piece of C++ code for doing this is attached at the end of this document, together with the sample input file ‘famot.xyz’. This program simply outputs the mapping "original atom number → new atom number". Here is the result of the renumbering.



1	S11	-3.15763	0.38730	4.11686	2	3	0	0
2	C10	-1.83328	-0.83920	4.04576	1	4	5	6
3	C12	-2.46400	1.75399	3.11132	1	20	21	22
4	C9	-1.92778	-1.82462	2.87380	2	7	8	9
5	H29	-1.88968	-1.31614	4.82938	2	0	0	0
6	H30	-1.03619	-0.39013	4.03420	2	0	0	0
7	C5	-1.67657	-1.14553	1.55244	4	10	11	0
8	H31	-1.26103	-2.49122	2.97977	4	0	0	0
9	H32	-2.82899	-2.21138	2.84944	4	0	0	0
10	N4	-0.40765	-0.88451	1.28417	7	12	0	0
11	N6	-2.72661	-0.84659	0.82513	7	18	19	0
12	S1	0.00000	0.00000	0.00000	10	13	14	15
13	O2	1.43958	0.00000	0.00000	12	0	0	0
14	O3	-0.66320	1.28419	0.00000	12	0	0	0
15	N33	-0.58734	-0.79346	-1.29434	12	16	17	0
16	H34	-0.72390	-0.27427	-1.94887	15	0	0	0
17	H35	-0.11417	-1.48289	-1.51453	15	0	0	0
18	H7	-2.64780	-0.44281	0.13183	11	0	0	0
19	H8	-3.51664	-1.07174	1.09032	11	0	0	0
20	C13	-1.37355	2.49913	3.79836	3	23	24	0
21	H27	-2.14046	1.40036	2.24489	3	0	0	0
22	H28	-3.18891	2.34331	2.91616	3	0	0	0
23	N14	-1.72206	3.44723	4.74729	20	25	0	0
24	C17	-0.04407	2.35168	3.57109	20	26	28	0
25	C15	-0.65834	4.05997	5.22388	23	26	27	0
26	S16	0.84898	3.46491	4.54237	24	25	0	0
27	N19	-0.55717	5.05435	6.13996	25	29	0	0
28	H18	0.39441	1.77470	2.93215	24	0	0	0
29	C20	-1.65577	5.66504	6.58976	27	30	31	0
30	N21	-1.49335	6.65177	7.47900	29	32	33	0
31	N24	-2.90102	5.36593	6.21922	29	34	35	0
32	H22	-0.74545	6.92387	7.68745	30	0	0	0
33	H23	-2.15564	7.03585	7.73829	30	0	0	0
34	H25	-3.02436	4.74583	5.69903	31	0	0	0
35	H26	-3.51859	5.80870	6.51171	31	0	0	0

Now we remove any *forward* references to atoms removed, as specified in 'Step 3' earlier, to give:

1	S11	-3.15763	0.38730	4.11686	
2	C10	-1.83328	-0.83920	4.04576	1
3	C12	-2.46400	1.75399	3.11132	1
4	C9	-1.92778	-1.82462	2.87380	2
5	H29	-1.88968	-1.31614	4.82938	2
6	H30	-1.03619	-0.39013	4.03420	2
7	C5	-1.67657	-1.14553	1.55244	4
8	H31	-1.26103	-2.49122	2.97977	4
9	H32	-2.82899	-2.21138	2.84944	4
10	N4	-0.40765	-0.88451	1.28417	7

11	N6	-2.72661	-0.84659	0.82513	7
12	S1	0.00000	0.00000	0.00000	10
13	O2	1.43958	0.00000	0.00000	12
14	O3	-0.66320	1.28419	0.00000	12
15	N33	-0.58734	-0.79346	-1.29434	12
16	H34	-0.72390	-0.27427	-1.94887	15
17	H35	-0.11417	-1.48289	-1.51453	15
18	H7	-2.64780	-0.44281	0.13183	11
19	H8	-3.51664	-1.07174	1.09032	11
20	C13	-1.37355	2.49913	3.79836	3
21	H27	-2.14046	1.40036	2.24489	3
22	H28	-3.18891	2.34331	2.91616	3
23	N14	-1.72206	3.44723	4.74729	20
24	C17	-0.04407	2.35168	3.57109	20
25	C15	-0.65834	4.05997	5.22388	23
26	S16	0.84898	3.46491	4.54237	24 25
27	N19	-0.55717	5.05435	6.13996	25
28	H18	0.39441	1.77470	2.93215	24
29	C20	-1.65577	5.66504	6.58976	27
30	N21	-1.49335	6.65177	7.47900	29
31	N24	-2.90102	5.36593	6.21922	29
32	H22	-0.74545	6.92387	7.68745	30
33	H23	-2.15564	7.03585	7.73829	30
34	H25	-3.02436	4.74583	5.69903	31
35	H26	-3.51859	5.80870	6.51171	31

Note that only a single atom has two connected atoms coming from it. This indicates that there is a *single* ring system in the molecule.

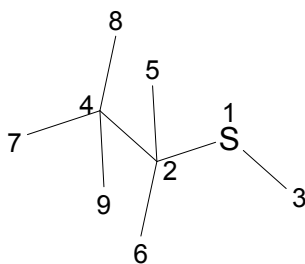
Famotidine: constructing the Z-matrix

Construction of the Z-matrix is now simply a matter of filling in the blanks in the table below. Atom 1 lies at 0,0,0, atom 2 lies at some distance a from atom 1, atom 3 lies at some distance b from atom 1 and makes an angle c with 1 and 2. Atom 4 lies at some distance d from atom 2, making an angle e with 2 and 1 and a torsion f with 2,1 and 3. This torsion can be varied. NB: We already have the entire first column (Bond to) in the form of the first connectivity column in the renumbered XYZ file.

#	Atom	Length	Angle	Torsion	Variable	Bond to	Angle to	Torsion to
1	S11	0.0	0.0	0.0	N	0	0	0
2	C10	a	0.0	0.0	N	1	0	0
3	C12	b	c	0.0	N	1	2	0
4	C9	d	e	f	Y	2	1	3
5	etc.....							

The calculation of distances, angles and torsions for the Z-matrix is straightforward in Cartesian space. By placing the blank Z-matrix next to the renumbered XYZ file, one can easily see the correspondence between them, and how the Z-matrix file can be generated automatically, using the bond connection information. All we need to remember for the moment is that each time we look back for a connection, we always use the connected atom with the lowest index number, of course remembering that we must exclude the atoms we are tracking from. For example, working through the first few atoms...

1	S11	-3.15763	0.38730	4.11686	
2	C10	-1.83328	-0.83920	4.04576	1
3	C12	-2.46400	1.75399	3.11132	1
4	C9	-1.92778	-1.82462	2.87380	2
5	H29	-1.88968	-1.31614	4.82938	2
6	H30	-1.03619	-0.39013	4.03420	2
7	C5	-1.67657	-1.14553	1.55244	4
8	H31	-1.26103	-2.49122	2.97977	4
9	H32	-2.82899	-2.21138	2.84944	4
...					
20	C13	-1.37355	2.49913	3.79836	3



1 lies at the origin. 2 is connected to 1. 3 is connected to 1 and as 1 is connected to 2, the angle is 3-1-2. 4 is connected to 2, 2 is connected to 1, 1 is connected to 3, so the angle must be 4-2-1 and the torsion 4-2-1-3. Torsion 4-2-1-3 must be flagged as a variable torsion. 5 is connected to 2, the lowest index connected to 2 is 1, and the lowest (excluding 2 as it has just been used) connected to 1 is 3. Therefore the torsion is 5-2-1-3. However, the central atoms 2-1 already participate in a rotating torsion and so atom 3 is replaced by the first atom that crossed the 2-1 bridge i.e. 4. The torsion thus becomes 5-2-1-4 i.e. an *improper* torsion that is *fixed* relative to the first atom of the *proper* torsion. Similarly, 6-2-1-3 must become 6-2-1-4. 7 is connected to 4, the lowest index from 4 is 2, the lowest index from 2 is 1, thus 7-4-2-1 is the correct torsion. This must be flagged as being variable. 8 is connected to 4, the lowest index from 4 is 2, the lowest index from 2 is 1, thus 8-4-2-1 is the suggested torsion. As the 4-2 bridge has already been crossed, the correct torsion must be 8-4-2-7. Similarly, 9-4-2-7 is correct torsion for atom 9. By the time we arrive at atom 20, we are starting to build the other side of the molecule but the same rules still apply. 20 is connected to 3, 3 to 1 and 1 to 2. The torsion is therefore 20-3-1-2.

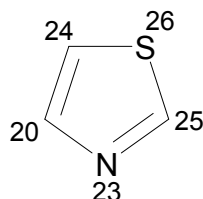
It should be clear by now that the rules for the construction of the Z-matrix become pretty straightforward once the renumbering process has been carried out. However, we still have a problem to deal with when handling ring systems.

Ring system handling

As mentioned earlier, the presence of a ring is indicated by an atom in the renumbered list having more than one connected atom. We need to identify the ring torsions and flag them to say that they cannot be variable. Looking more closely at atom 26,

20	C13	-1.37355	2.49913	3.79836	3
21	H27	-2.14046	1.40036	2.24489	3
22	H28	-3.18891	2.34331	2.91616	3
23	N14	-1.72206	3.44723	4.74729	20
24	C17	-0.04407	2.35168	3.57109	20
25	C15	-0.65834	4.05997	5.22388	23
26	S16	0.84898	3.46491	4.54237	24 25

we see that it has more than one connected atom. By following the connections from these connected atoms, we see that the ring must consist of



and so any torsion that involves a central bond of 26-24, 26-25, 24-20, 25-23 or 23-20 cannot be variable. To illustrate that this works, the rules outlined thus far would generate the following torsions leading up to and around the ring.

20-3-1-2	Variable Proper
21-3-1-20	Fixed Improper
22-3-1-20	Fixed Improper
23-20-3-1	Variable Proper
24-20-3-23	Fixed Improper
25-23-20-3	Fixed Proper (because it involves a central bond of 23-20)
26-24-20-3	Fixed Proper (because it involves a central bond of 24-20)
27-25-23-20	Fixed Proper (because it involves a central bond of 25-23)
28-24-20-3	Fixed Proper (because it involves a central bond of 24-20)

Remember that the distinction between a proper and an improper torsion is one of nomenclature and the only difference that we actually see in the final Z-matrix between these different types of torsions is: are they flagged as being variable or not?

Connectivity in the final Z-matrix as a result of applying the rules described is given below, with actual lengths, angles and torsions omitted for clarity.

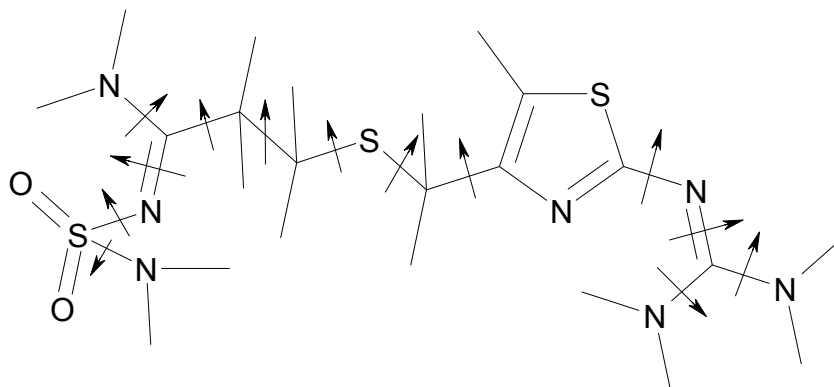
#	Atom	Len	Ang	Tor	Var	Bond to	Angle to	Torsion to
1	S11				N	0	0	0
2	C10				N	1	0	0
3	C12				N	1	2	0
4	C9				Y	2	1	3
5	H29				N	2	1	4
6	H30				N	2	1	4
7	C5				Y	4	2	1
8	H31				N	4	2	7
9	H32				N	4	2	7
10	N4				Y	7	4	2
11	N6				N	7	4	10
12	S1				Y ^{db}	10	7	4
13	O2				Y	12	10	7
14	O3				N	12	10	13
15	N33				N	12	10	13
16	H34				Y ^h	15	12	10
17	H35				N	15	12	16
18	H7				Y ^h	11	7	4
19	H8				N	11	7	18
20	C13				Y	3	1	2
21	H27				N	3	1	20
22	H28				N	3	1	20
23	N14				Y	20	3	1
24	C17				N	20	3	23
25	C15				N ^r	23	20	3
26	S16				N ^r	24	20	3
27	N19				N ^r	25	23	20
28	H18				N ^r	24	20	22
29	C20				Y	27	25	23
30	N21				Y ^{db}	29	27	25
31	N24				N	29	27	30
32	H22				Y ^h	30	29	27
33	H23				N	30	29	32
34	H25				Y ^h	31	29	27
35	H26				N	31	29	34

^r indicates ring torsion

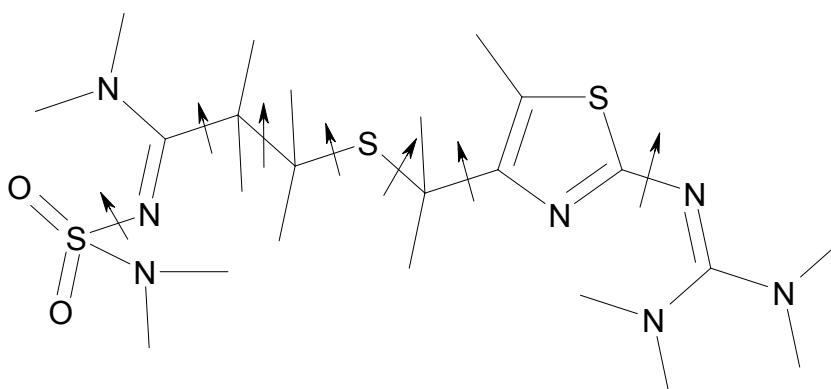
^h indicates torsion that will twist only H atoms and so is not really needed

^{db} indicates torsion around a double bond that is not needed

Therefore, the algorithm suggests a total of 13 torsions, which a glance at a sketch of the molecule shows to be correct.



However, applying our chemical knowledge (eliminating double bonds and rotations that only affect the positions of H atoms), we see that only 7 make any sense for an X-ray powder diffraction experiment



Summary

The algorithm outlined above, as implemented in the DASH computer program for structure determination from powder diffraction data, has proven to be very successful. The critical step is the renumbering of the structure; once this is done, generation of the Z-matrix is straightforward. Although alternatives to the Z-matrix approach exist, it remains a very simple and useful formalism.

1. Shankland, K. (2004). <http://www.iucr.org/iucr-top/comm/ccom/newsletters/2004aug/>
2. Shankland, K., McBride, L., David, W.I.F., Shankland, N., Steele, G., (2002) *J. Appl. Crystallogr.*, **35**, 443-454.

Atom renumbering code (reads file 'famot.xyz')

```
#include <iostream.h>
#include <fstream.h>
#include <math.h>
int used[100], ic[100][5], ic2[100][5], numlines;
double xyz[100][3], xyz2[100][6];
void trace(int i);

void main() {
    int j, junk, itrace;
    char junk2[2];
    ifstream infile("famot.xyz",ios::in);
    infile >> numlines;
    for (j=1; j<=numlines; j++)
        infile >>junk>>junk2>>xyz[j][1]>>xyz[j][2]
        >>xyz[j][3]>>ic[j][1]>>ic[j][2]>>ic[j][3]>>ic[j][4];
    for (j=0; j<=11; j++)
        used[j] = 0;
    cout << "trace from which atom number ? ";
    cin >> itrace;
    trace(itrace);
}

void trace(int i) {
    static int incr=0, mapping[100], traced[100];
    int j, k;
    traced[i]=1;
    if (used[i] == 0) {
        used[i] = 1;
        incr++;
        mapping[i]=incr;
        xyz2[mapping[i]][1]=xyz[i][1];
        xyz2[mapping[i]][2]=xyz[i][2];
        xyz2[mapping[i]][3]=xyz[i][3];
        ic2[mapping[i]][1]=ic[i][1];
        ic2[mapping[i]][2]=ic[i][2];
        ic2[mapping[i]][3]=ic[i][3];
        ic2[mapping[i]][4]=ic[i][4];
        cout << "mapping " << i << " to " << mapping[i] << endl;
    }

    for (j=1; j<=4; j++) {
        if (ic[i][j] != 0) {
            if (used[ic[i][j]] == 0) {
                used[ic[i][j]] = 1;
                incr++;
                mapping[ic[i][j]]=incr;
                xyz2[mapping[i]][1]=xyz[i][1];
                xyz2[mapping[i]][2]=xyz[i][2];
                xyz2[mapping[i]][3]=xyz[i][3];
                ic2[mapping[i]][1]=ic[i][1];
                ic2[mapping[i]][2]=ic[i][2];
                ic2[mapping[i]][3]=ic[i][3];
                ic2[mapping[i]][4]=ic[i][4];
                cout << "mapping " << ic[i][j] << " to " << mapping[ic[i][j]] << endl;
            }
        }
    }

    for (j=1; j<=4; j++) {
        k=ic[i][j];
        if ( (k!=0) && (traced[k]==0)) {
            trace(k);
        }
    }
}
```

35							
1	S1	0.00000	0.00000	0.00000	2	3	4 33
2	O2	1.43958	0.00000	0.00000	1	0	0 0
3	O3	-0.66320	1.28419	0.00000	1	0	0 0
4	N4	-0.40765	-0.88451	1.28417	1	5	0 0
5	C5	-1.67657	-1.14553	1.55244	4	6	9 0
6	N6	-2.72661	-0.84659	0.82513	5	7	8 0
7	H7	-2.64780	-0.44281	0.13183	6	0	0 0
8	H8	-3.51664	-1.07174	1.09032	6	0	0 0
9	C9	-1.92778	-1.82462	2.87380	5	10	31 32
10	C10	-1.83328	-0.83920	4.04576	9	11	29 30
11	S11	-3.15763	0.38730	4.11686	10	12	0 0
12	C12	-2.46400	1.75399	3.11132	11	13	27 28
13	C13	-1.37355	2.49913	3.79836	12	14	17 0
14	N14	-1.72206	3.44723	4.74729	13	15	0 0
15	C15	-0.65834	4.05997	5.22388	14	16	19 0
16	S16	0.84898	3.46491	4.54237	15	17	0 0
17	C17	-0.04407	2.35168	3.57109	13	16	18 0
18	H18	0.39441	1.77470	2.93215	17	0	0 0
19	N19	-0.55717	5.05435	6.13996	15	20	0 0
20	C20	-1.65577	5.66504	6.58976	19	21	24 0
21	N21	-1.49335	6.65177	7.47900	20	22	23 0
22	H22	-0.74545	6.92387	7.68745	21	0	0 0
23	H23	-2.15564	7.03585	7.73829	21	0	0 0
24	N24	-2.90102	5.36593	6.21922	20	25	26 0
25	H25	-3.02436	4.74583	5.69903	24	0	0 0
26	H26	-3.51859	5.80870	6.51171	24	0	0 0
27	H27	-2.14046	1.40036	2.24489	12	0	0 0
28	H28	-3.18891	2.34331	2.91616	12	0	0 0
29	H29	-1.88968	-1.31614	4.82938	10	0	0 0
30	H30	-1.03619	-0.39013	4.03420	10	0	0 0
31	H31	-1.26103	-2.49122	2.97977	9	0	0 0
32	H32	-2.82899	-2.21138	2.84944	9	0	0 0
33	N33	-0.58734	-0.79346	-1.29434	1	34	35 0
34	H34	-0.72390	-0.27427	-1.94887	33	0	0 0
35	H35	-0.11417	-1.48289	-1.51453	33	0	0 0

Call for Contributions to the Next CompComm Newsletter

Due to the IUCr 2005 Florence congress occurring in the middle of the year, the sixth issue of the Compcomm Newsletter is expected to appear around January of 2006 with the primary theme to be determined. If no-one is else is co-opted, the newsletter will be edited by Lachlan Cranswick.

Contributions would be also greatly appreciated on matters of general interest to the crystallographic computing community, e.g. meeting reports, future meetings, developments in software, algorithms, coding, programming languages, techniques and other news.

Please send articles and suggestions directly to the editor.

Lachlan M. D. Cranswick

NPMR, NRC,

Building 459, Station 18,

Chalk River Laboratories,

Chalk River, Ontario,

Canada, K0J 1J0

E-mail: lachlan.cranswick@nrc.gc.ca

WWW: <http://neutron.nrc.gc.ca/peep.html#cranswick>