



# Commission on Crystallographic Computing International Union of Crystallography

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

Newsletter No. 6, September 2005

**This issue:**

**"Talk notes from the IUCr Computing School,  
Siena, Italy, 18th to 23rd August 2005"**

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

## Table of Contents (page 1 of 2)

(Notes collated by Lachlan Cranswick)

(Warning – unless you want to kill 323 pages worth of forest – DO NOT press the “print” button. For hardcopies – you may like to only print out the articles of personal interest.)

<b>CompComm chairman's message, <i>Ton Spek</i></b>	3	<b>GUI Design</b>	76
<b>Newsletter No. 6, <i>Lachlan Cranswick</i></b>	3	<i>Brian H. Toby</i>	
<b>IUCr Commission on Crystallographic Computing</b>	4	<b>Automated data collection and integration</b>	82
		<i>Rob Hooft</i>	
<b>Participants of the IUCr Computing School, Certosa di Pontignano, University of Siena, Tuscany, Italy, Thursday 18th - Tuesday 23rd August 2005.</b>	5	<b>Operating Hardware</b>	95
		<i>Rob Hooft</i>	
<b>Siena 2005 IUCr Computing School Talk Notes :</b>		<b>Integration of 2D diffraction images</b>	100
<b>Introductory talk</b>	6	<i>James W. Pflugrath</i>	
<i>Ton Spek</i>		<b>Atomic pair distribution function (PDF) analysis: What, When, Why, How?</b>	107
<b>Porting between Operating Systems</b>	12	<i>Simon J.L. Billinge</i>	
<i>Harry Powell</i>		<b>Dealing with overlapped data</b>	116
<b>Modern approaches to programming</b>	18	<i>Bill David</i>	
<i>Ralf Grosse-Kunstleve</i>		<b>Programming the Science of Crystallography</b>	122
<b>Using available tools</b>	23	<i>Ton Spek</i>	
<i>Louis Farrugia</i>		<b>Simple algorithms for macromolecular phasing</b>	129
<b>Scripting languages / Spectrum of languages</b>	30	<i>George M. Sheldrick</i>	
<i>Ralf Grosse-Kunstleve</i>		<b>Automation of structure determination</b>	133
<b>Legacy Codes. Do They Have Any Value?</b>	32	<i>Tom Terwilliger</i>	
<i>David Watkin</i>		<b>Crystallographic Symmetry in Real and Reciprocal Space</b>	140
<b>Complete rewrites. When, why, and how?</b>	47	<i>Kevin Cowtan</i>	
<i>James W. Pflugrath</i>		<b>Profile refinement Least-squares analysis and beyond</b>	147
<b>Coordinate systems, operators, and transformations</b>	52	<i>Bill David</i>	
<i>Kevin Cowtan</i>		<b>Fourier Transforms in Crystallography</b>	156
<b>"New" algorithms</b>	60	<i>Lynn ten Eyck</i>	
<i>Tom Terwilliger</i>		<b>Maximum Likelihood in X-ray Crystallography</b>	160
<b>Connecting programs together</b>	67	<i>Kevin Cowtan</i>	
<i>Louis Farrugia</i>		<b>Overview of Crystallographic Structure Refinement</b>	166
<b>Program Suites</b>	71	<i>Lynn ten Eyck</i>	
<i>Harry Powell</i>			

<b>Refinement II - Modern Developments</b>	168		
<i>Dale E. Tronrud</i>			
<b>CCP14 workshop Notes: On Minimization Targets and Algorithms</b>	171	<b>CrysFML: A crystallographic library in modern Fortran</b>	271
<i>Dale E. Tronrud</i>		<i>Juan Rodríguez-Carvajal</i>	
<b>Computational aspects of the Rietveld Method</b>	176	<b>Connecting to hardware</b>	280
<i>Juan Rodríguez-Carvajal</i>		<i>Rob Hooft</i>	
<b>Statistical Treatment of Uncertainties</b>	183	<b>Tcl/Tk demo</b>	286
<i>Dale E. Tronrud</i>		<i>Brian Toby</i>	
<b>Programming pdCIF and Rietveld</b>	187	<b>CCTBX Unit Cell Refinement Tutorial</b>	292
<i>Brian H. Toby</i>		<i>Ralf W. Grosse-Kunstleve</i>	
<b>Structure Comparison, Analysis and Validation</b>	194	<b>CCTBX Direct Methods Tutorial</b>	303
<i>Ton Spek</i>		<i>Ralf W. Grosse-Kunstleve</i>	
<b>Testing software</b>	202	<b>CCTBX Central Types</b>	317
<i>Harry Powell</i>		<i>Ralf W. Grosse-Kunstleve</i>	
<b>The future of direct methods</b>	206	<b>Tutorial on scoring methods for evaluation of electron density maps</b>	318
<i>George M. Sheldrick</i>		<i>Tom Terwilliger</i>	
<hr/>		<hr/>	
<b>Siena 2005 IUCr Computing School Tutorials :</b>		<b>Calls for contributions to Newsletter No. 7</b>	323
<b>Using the Clipper libraries</b>	210		
<i>Kevin Cowtan</i>			
<b>Exercises in map manipulation</b>	220		
<i>Kevin Cowtan</i>			
<b>Reciprocal Space Tutorial (RST)</b>	224		
<i>George M. Sheldrick</i>			
<b>Fortran 77 solution to GMS RST</b>	229		
<i>George M. Sheldrick</i>			
<b>C++ solution to GMS RST</b>	232		
<i>Tim Gruene</i>			
<b>C++ solution to GMS RST</b>	239		
<i>Michel Fodje</i>			
<b>Fortran 95 solution to GMS RST</b>	247		
<i>Juan Rodríguez-Carvajal</i>			
<b>Java solution to GMS RST</b>	252		
<i>Bradley Smith</i>			
<b>Fortran solution to GMS RST</b>	258		
<i>Stephan Ruehl</i>			
<b>Initial Python solution to GMS RST</b>	269		
<i>Ralf W. Grosse-Kunstleve</i>			
<b>Slightly Enhanced Python solution to GMS RST</b>	270		
<i>Ralf W. Grosse-Kunstleve</i>			



---

## CompComm Chairman's Message

This newsletter makes the presentations available to a wider audience of the lectures given at the IUCr Crystallographic Computing School 2005 in Siena, prior to the Florence 2005 IUCr congress. This breaks with a tradition of the past where, based on the material presented, a very expensive book was produced with rather limited distribution. We hope that the current approach that we have chosen for the distribution of the lecture material fits more in our 'Google-age', where a lot of information is now looked for on the WEB.

One of the implicit themes during this meeting was the software language to be used for future software development. Traditionally, Fortran in its various incarnations was the default language. A large number of programs that are in use today are written in that language (e.g. the SHELX suite). Fortran could still be the language of choice today for scientific computing in view of its simplicity and efficiency. However, computing science has now moved to C-type and scripting languages, of which C++ and Python are currently the most popular respectively. A large number of libraries for graphical user interfaces are currently available for those languages. C++ will be the de-facto language of the future, notwithstanding its sharp learning curve. Interestingly, George Sheldrick (one of the lecturers and a longstanding Fortran addict like me) has now taught himself C++ as an outcome of this very successful school.

I would like to thank the speakers again for all the work they did to prepare their presentations, the sponsors for their financial support, my co-organisers, the University of Siena for making available an excellent meeting site and Prof. Marcello Mellini for making it all work locally.

*Ton Spek, Chairman of the IUCr computing school and IUCr Computing Commission,  
([a.l.spek@chem.uu.nl](mailto:a.l.spek@chem.uu.nl))*

---

## Newsletter No. 6

This newsletter collates the notes from the Siena 2005 IUCr Crystallographic Computing School. The individual files are located on the school webpage at:

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

The next edition of the newsletter is expected to get back to the usual format with an intended theme involving "Minimisation" algorithms for indexing, structure solution and refinement. It should hopefully appear around April 2006.

*Lachlan Cranswick,  
([lachlan.cranswick@nrc.gc.ca](mailto:lachlan.cranswick@nrc.gc.ca))*

---

## THE IUCR COMMISSION ON CRYSTALLOGRAPHIC COMPUTING - TRIENNium 2005-2008

---

**Chairman: Professor 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](mailto:a.l.spek@chem.uu.nl)  
WWW: <http://www.cryst.chem.uu.nl/spea.html>

**Lachlan M. D. Cranswick**

Canadian Neutron Beam Centre (CNBC),  
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](mailto:lachlan.cranswick@nrc.gc.ca)  
WWW: <http://neutron.nrc.gc.ca/peep.html#cranswick>

**Dr Ralf W. Grosse-Kunstleve**

Lawrence Berkeley National Laboratory  
One Cyclotron Road, BLDG 64R0121,  
Berkeley, California, 94720-8118, USA.  
Tel: (510) 486-5929  
Fax: (510) 486-5909  
E-mail: [RWGrosse-Kunstleve@lbl.gov](mailto:RWGrosse-Kunstleve@lbl.gov)  
WWW: <http://cci.lbl.gov/>

**Professor 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](mailto:alex@unimore.it)  
WWW: <http://www.terra.unimo.it/mineralogia/gualtieri.html>

**Professor Luhua Lai**

Institute of Physical Chemistry,  
Peking University,  
Beijing 100871,  
China.  
Fax: +86-10-62751725.  
E-mail: [lh lai@pku.edu.cn](mailto:lh lai@pku.edu.cn)  
WWW: <http://mdl.ipc.pku.edu.cn/>

**Dr Airlie McCoy**

Structural Medicine,  
Cambridge Institute for Medical Research (CIMR),  
Wellcome Trust/MRC Building,  
Addenbrooke's Hospital, Hills Road, Cambridge CB2 2XY, UK  
Tel: +44 (0) 1223 217124  
Fax: +44 (0) 1223 217017  
E-mail: [ajm201@cam.ac.uk](mailto:ajm201@cam.ac.uk)  
WWW: <http://www-structmed.cimr.cam.ac.uk/>

**Professor Atsushi Nakagawa**

Research Center for Structural and Functional Proteomics,  
Institute for Protein Research,  
Osaka University,  
3-2 Yamadaoka, Suita,  
Osaka, 565-0871, Japan  
Tel: +81-(0)6-6879-4313  
Fax: +81-(0)6-6879-4313  
E-mail: [atsushi@protein.osaka-u.ac.jp](mailto:atsushi@protein.osaka-u.ac.jp)  
WWW: <http://www.protein.osaka-u.ac.jp/>

**Dr. Simon Parsons**

School of Chemistry, University of Edinburgh  
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](mailto:s.parsons@ed.ac.uk)  
WWW: <http://www.chem.ed.ac.uk/staff/parsons.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](mailto:harry@mrc-lmb.cam.ac.uk)  
WWW: <http://www.mrc-lmb.cam.ac.uk/harry/>

**Consultants****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: [idbrown@mcmaster.ca](mailto:idbrown@mcmaster.ca)  
WWW: [http://www.physics.mcmaster.ca/people/faculty/Brown\\_ID\\_h.html](http://www.physics.mcmaster.ca/people/faculty/Brown_ID_h.html)

**Professor Eleanor Dodson**

York Structural Biology Laboratory,  
Department Of Chemistry,  
University of York, Heslington,  
York, YO10 5YW, UK  
Tel: +44 (1904) 328253  
Fax: +44 1904 328266  
E-mail: [e.dodson@ysbl.york.ac.uk](mailto:e.dodson@ysbl.york.ac.uk)  
WWW: <http://www.ysbl.york.ac.uk/people/6.htm>

**Dr David Watkin**

Chemical Crystallography, Department of Chemistry,  
University of Oxford, Chemistry Research Laboratory,  
Mansfield Road,  
Oxford, OX1 3TA, UK  
Tel: +44 (0) 1865 285019  
Fax: +44 (0) 1865 285021  
E-mail: [david.watkin@chemistry.oxford.ac.uk](mailto:david.watkin@chemistry.oxford.ac.uk)  
WWW: <http://www.chem.ox.ac.uk/researchguide/djwatkin.html>

---

## Participants of the IUCr Computing School, Certosa di Pontignano, University of Siena, Tuscany, Italy, Thursday 18th - Tuesday 23rd August 2005.



Organisers: Ton Spek, Marcello Mellini, Alessandro Gualtieri, Harry Powell, Lachlan Cranswick  
Consultants: Simon Parsons, David Watkin

IUCr Representative: Davide Viterbo

Lecturers: Simon Billinge, Kevin Cowtan, Bill David, Louis Farrugia, Ralf Grosse-Kunstleve, Rob Hooft, James Pfugrath, Harry Powell, Juan Rodríguez-Carvajal, George Sheldrick, Ton Spek, Lynn ten Eyck, Tom Terwilliger, Brian Toby, Dale Tronrud, David Watkin

Registrants: Priyamvada Acharya, Sandor Brockhauser, Jeppe Christensen, Jeremy Cockcroft, Fabio Dall'Antonia, Paul Emsley, ZhenJie Feng, Laura Rocas Fernández, Laura Torre Fernández, Marc Fleurant, Michel Fodje, Frantisek Frantisek, Nicholas Furnham, Roni Gordon, Tim Gruene, Carmen Guguta, Ilia Guzei, Svetlana Ivashevskaya, Soorya N Kabekkodu, Huub Kooijman, Gert J. Kruger, Maciej Kubicki, Martin Lutz, Yvonne P. Mascarenhas, Ernesto Mesto, Alexander Nazarenko, Steven Ness, Aritra Pal, Pryank Patel, Simon Parsons, Mike Probert, Cristy Leonor Azanza Ricardo, Stephan Ruehl, Irakli Sikharulidze, Pavol Skubak, Alexander B. Smith, Bradley Smith, Margareta Sorenson, Richard Stephenson, Gudrun Stranzl, Pavel Teslenko, Michael Turner, Peter Turner, Marco Voltolini, Marcin Wojdyr, Pete Wood, Jon Wright, Xinyi Xian,

Accompanying Persons: Linda ten Eyck, Aviva Pelt, Katherine Sheldrick

Sponsors:

Università di Siena  
MAX-INF2  
Merck & Co., Inc. - USA  
Bruker AXS  
Cambridge Crystallographic Data Centre (CCDC)  
CCP4 - Collaborative Computing Project Number 4 in Protein Crystallography  
Oxford Diffraction  
Rigaku/MSD  
International Union of Crystallography (IUCr)

## INTRODUCTORY TALK

Ton Spek  
National Single Crystal Service  
Facility  
Utrecht University

### Not so 40 Years ago

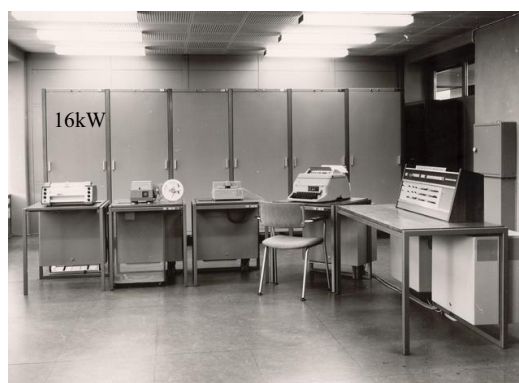
- The crystallography group in Utrecht already had a tradition in Direct Methods (Paul Beurskens, one of the first authors implementing the *Symbolic Addition Method*). However, none of the locally available programs gave an interpretable hand-contoured map.
- So I ended up with developing my own *Symbolic Addition* program, AUDICE, for centro-symmetric structures.
- AUDICE was locally rather successful since it also solved all other notoriously 'unsolvable structures' hanging around in the lab.
- Major calculations and program testing were done once a week during the 'nightshift', not 'overnight', on the Utrecht University mainframe. (A major social event in those days in view of the presence of most group members between 6PM and 8 AM the next morning)



Flexowriter for the creation and editing of programs and data

### Some History

- On crystallographic computing from the perspective of a small-molecule crystallographer who started to work in crystallography in the mid 60's at Utrecht University, The Netherlands.
- Many of the older software developers, like me, have a background in Direct Methods. Mine started as follows:
- As a student, I was given a colorless crystal of unknown composition with the assignment to determine its structure using X-ray techniques only. It took me more than ½ a year to determine that it was *methoxyglutaconic acid*.
- Today, 40 years later, a problem like this is solved in a matter of seconds on my notebook, but not in those days.



~1966, Electrologica X8 ALGOL60 'Mainframe' (<1MHz)

### Times and Mainframe Changed

- **MULTAN (FORTRAN/PUNCHCARD)** came and replaced my Direct Methods program AUDICE (ALGOL60/Papertape) in the early 70's, when the single user university computer was replaced by a real multiuser mainframe (CDC6400).
- MULTAN was superseded in the 80's by the even more powerful **SHELXS, SIR & DIRDIF** software.
- No big improvements in small molecule DM since then ?
- In the 90's **S&B, SHELXD** entered the field, coming down from Macro-crystallography.



## Direct Methods Meetings

- Many past meetings and schools were organized with Direct Methods (software and theory) as a major subject.
- Important one's were the CECAM workshops on Direct Methods (5 weeks!, bringing together people working in the field to work on current issues) in the early 70's in Orsay around a big IBM-360 with lectures by Hauptman.
- Launch of MULTAN, many personal contacts – Viterbo
- NATO schools on Direct Methods in Parma and York in the 70's.
- Direct Methods schools in Erice in 1974 & 1978.
- Photo of the participants of the 1978 Erice School next :

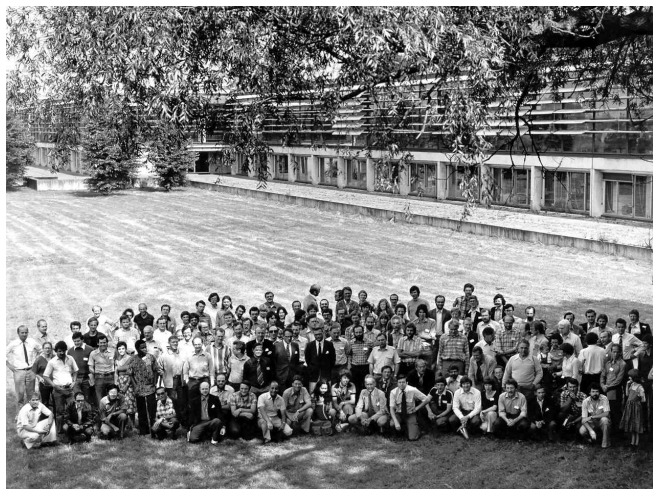


## Direct Methods Now

- Direct Methods appear to be currently no longer a major topic at meetings.
- Some years ago there was a morning lecture by Herbert Hauptman with the message that the tangent formula was what really mattered. That afternoon there was a lecture by Carmello Giacovazzo with the message that there was no need for the tangent formula...
- George Sheldrick will give us his perspective on '*The Future of Direct Methods*' at the end of this meeting.

## IUCr Computing Schools

- ..... Mostly held jointly with IUCr Assemblies – Examples
- 1963 – Rollett, Algorithms (black book)
- 1969 - Least-Squares & Absorption Correction (SHELX76 - code)
- 1978 - Program systems (SHELX, XTAL, NRCVAX etc.)
- 1996 - Macro-crystallography
- 1999 - Macro-crystallography
- 2002 – (None)
- 2005 - Siena (again: Small, Macro, Powder)
- Photo 1978 school in Enschede (Netherlands)



## Motivations for this Crystallographic Computing School

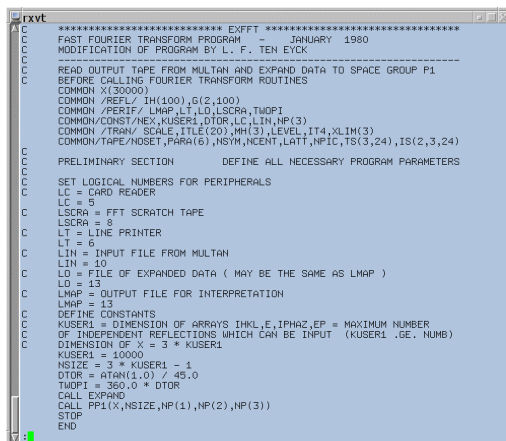
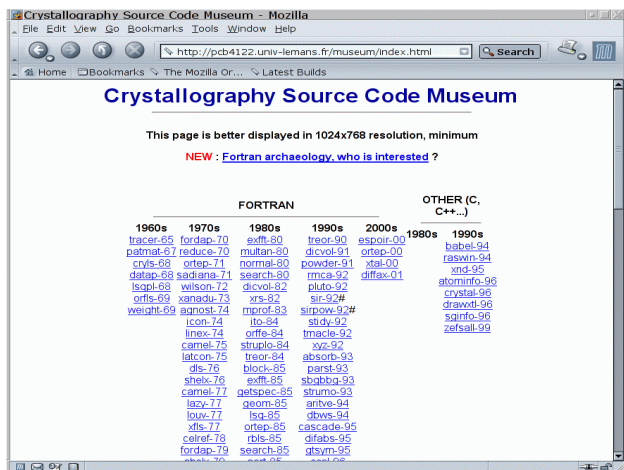
- A general feeling within at least the small-molecule community: *'The current generation of software developers is phasing out, where is the new generation to keep things running in the future'*
- There exists a growing community of push-button users (*What is not behind a button can not be done...*)
- Major funding and software development is currently in macro crystallography (*possible useful spin-off to the small-molecule world*)
- It is sensed that things well known in the small-molecule world are reinvented in the macro world and presented as new.
- Black Box and Proprietary Software as opposed to Open Source. (*lack of info about the algorithms used and options to modify*)

## Hardware Platforms

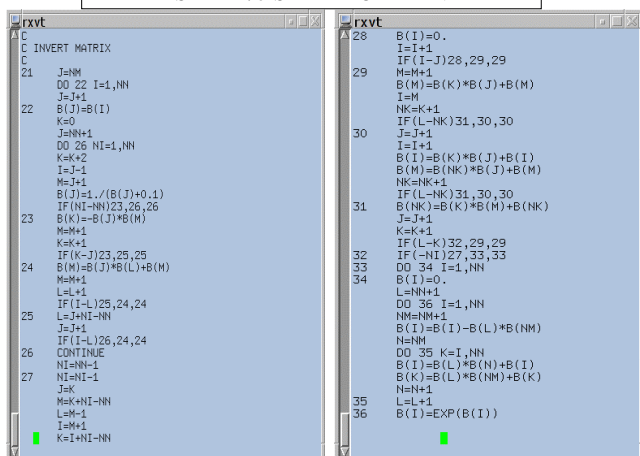
- MS-Windows:
  - Small-Molecule Crystallography
  - Powder crystallography
- UNIX/LINUX/(OSX):
  - Macro Crystallography
  - (Small-Molecule Crystallography)

## Software Languages

- Crystallographic software has been written in *machine language*, *assembly language*, *algol60*, *(turbo)basic*, *(turbo)pascal*, *Fortran*, *C*, *C++* and various scripting languages such as *python*
- 'Stone-age' *Fortran* based software is still ubiquitous in the small-molecule world (ORTEP, SHELX, CRYSTALS, PLATON etc.)
- New (commercial) software development mainly in C++ and scripting languages.
- A project just started in the UK to Rethink & Rewrite old Fortran based software to C++ (Durham, Oxford project).
- Old software saved in 'The Crystallographic Source Code Museum' by Armal LeBail, supposedly interesting to look for useful algorithms.



### SHELX76-STYLE FORTRAN



## Alternative Algorithms for the Implementation of the same Task

- Tasks can usually be programmed in a variety of ways with widely ranging claims on memory and CPU resources.
- It is important to know the actual application to make the relevant decisions.
- Following is a simple, though somewhat extreme, example from the 1960's where a theoretical idea in Direct Methods was given to a professional programmer to implement.
- The final program was nicely written and documented.
- However, the calculation didn't terminate within hours even for a trivial application ....(my mystery structure)

### Problem from Symbolic Addition Method

P+ for triple H,K,H+K depends on

$$|E(H)E(K)E(H+K)|$$

'Correlation Method' → Improved P+

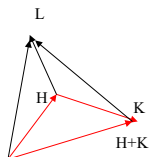
on the basis of P+ of three adjacent triples

$$|E(H)E(L)E(H+L)|$$

$$|E(K)E(L-K)E(L)|$$

$$|E(H+K)E(L-K)E(H+L)|$$

I.e. Strengthening of P+ ( $|E(H)E(K)E(H+K)|$ ) when in addition  $E(H+L), E(L-K), E(L)$  strong  
(Note: Theoretically formalized in terms of neighbourhoods, Hauptman)



## Two Implementations

### Implementation I: (Professional Programmer)

- 1 - Search and store all triple products found with  $E > E(\min)$
- 2 - Find from this list quartets of triples forming a tetrahedron

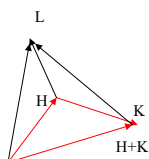
**Problem with 1:** The number of triplets explodes with increasing size of the structure at hand and so memory requirements (limited to 16kW in those days)

**Problem with 2:** Multiple nested loops with large range

### Implementation II (by Young Student)

Generate 'correlations' on the fly during triple relation search by looping with L with  $E(L) > E(\min)$  and testing for large  $E(L-K)$  and  $E(H+L)$ .

**Result:** Completion of the search in minutes rather than hours.



## Numerical Recipes

- An excellent and rich source of numerical routines for sorting, optimisation, FFT etc. with associated background is the book *Numerical Recipes* by W.H. Press et al., that has separate Fortran and C versions

## Numerical Recipe Example

- A very nice routine from NR is code with the name 'FOURN.FOR'.
- Forward and Backward FFT in N dimensions.
- In our crystallographic application:  $N = 3$
- Code = 69 Fortran lines ! Next ....

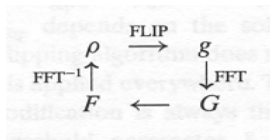
```

bea@xray2:~/sfun3/02/014/shelxs6/exor
Edit View Terminal Go Help
SUBROUTINE FOURN(DATA,NN,NDIM,ISIGN)
REAL*8 WR,WI,WPR,WPI,WTEMP,THETA
DIMENSION NN(NDIM),DATA(*)
NTOT=1
DO 11 IDIM=1,NDIM
  NTOT=NTOT*NN(IDIM)
CONTINUE
NPREV=1
DO 18 IDIM=1,NDIM
  N=NN(IDIM)
  NREM=NTOT/(N*NPREV)
  IP1=2*NPREV
  IP2=IP1*N
  IP3=IP2*NREM
  I2REV=1
  DO 14 I2=1,IP2,IP1
    IF(I2.LT.I2REV)THEN
      DO 13 I1=I2,I2+IP1-2,2
        DO 12 I3=I1,IP3,IP2
          I3REV=I2REV+I3-I2
          TEMPR=DATA(I3)
          TEMPI=DATA(I3+1)
          DATA(I3)=DATA(I3REV)
          DATA(I3+1)=DATA(I3REV+1)
          DATA(I3REV)=TEMPR
          DATA(I3REV+1)=TEMPI
        CONTINUE
      ENDFI
      IBIT=IP2/2
      IF ((IBIT.GE.IP1).AND.(I2REV.GT.IBIT)) THEN
        I2REV=I2REV-IBIT
        IBIT=IBIT/2
      GO TO 1
    ENDFI
    I2REV=I2REV+IBIT
  DO 15 I2=1,IP3,IP2
    K1=I2
    K2=K1+IP1
    TEMPR=SINGL(WR)*DATA(K2)-SINGL(WI)*DATA(K2+1)
    TEMPI=SINGL(WR)*DATA(K2+1)+SINGL(WI)*DATA(K2)
    DATA(K2)=DATA(K1)-TEMPR
    DATA(K2+1)=DATA(K1+1)-TEMPI
    DATA(K1)=DATA(K1)+TEMPR
    DATA(K1+1)=DATA(K1+1)+TEMPI
  CONTINUE
  WTEMP=WR
  WR=WI
  WI=WTEMP-WPR-WPI-WI
  WPR=WR+WPR-WI*WPI-WI
  WPI=WI+WPI-WTEMP*WPI-WI
CONTINUE
  IF(I2REV.GE.IP1).AND.(I2REV.GT.IBIT)) THEN
    I2REV=I2REV-IBIT
    IBIT=IBIT/2
  GO TO 1
ENDFI
I2REV=I2REV+IBIT
END

```

## Application of FOURN.FOR

- *Ab Initio* structure solution by charge flipping
- See G.Oszlanyi & A. Suto, (2004) Acta Cryst A60,134.
- Procedure: cycle between reciprocal space to direct space and back after modification of the density map until convergence using forward and backward FFT.



## So, No More D.M. ?

- Preliminary results on real structures, including incommensurate structures look interesting.
- There will be a lecture on this in Florence (MS20).
- Faster FFT: (Free C-library) FFTW  
However, with greater implementation and portability complexities.

## Other Computing Areas

- Powder (indexing, solution, refinement)
- RDF (Billinge)
- Macro Xtal (Phasing, Building, Refinement)
- Charge Density Studies (XD)
- Least Squares and other optimisation techniques.

## Other Computing Areas

- Incommensurate Structures (solution, refinement) (Keynote lecture by Petricek in Florence).
- Graphics (GUI's and presentation)
- Data collection and data reduction.
- Databases, Structure analysis and Validation

## The Program of the School

- There has been some discussion in the program commission on whether there should be two largely parallel sessions in view of a perceived growing diversion of interest.
- Eventually this path was not pursued, resulting in the current program that involves a mix of small-molecule, macro-molecule and powder interests.
- This format should provide a fruitful platform to pick up and discuss ideas from each others field.

## The Program of the School

- Lecturers were asked to focus on software development and internals rather than presenting the latest science or user instruction to their software.
- Not a school to learn basic programming.
- An introduction to current software development techniques (scripting languages, toolboxes etc.)
- Hands-on projects and workshops on personal notebooks.
- Bringing together representatives of the older and a next generation interested in software development.



Program Schedules Venue Registration Travel Accommodation Contacts Useful Links Student Bursaries Sponsors' Notes Sponsors Organisers School History Accompanying Persons 2005 Comm Comm School Logo		Thursday 18th August	Friday 19th August	Saturday 20th August	Sunday 21st August	Monday 22nd August	Tuesday 23rd August
	07:30 09:00		breakfast				
	09:45		<a href="#">modern approaches to programming</a> <i>Ralf Grosse-Kunstleve (Berkeley)</i>	<a href="#">intro to connecting programs together</a> <i>Louis Farrugia (Glasgow)</i>	<a href="#">dealing with overlapped data</a> <i>Bill David (RAL)</i>	<a href="#">Fourier methods</a> <i>Lynn ten Brink (San Diego)</i>	<a href="#">programming for CIF</a> <i>Brian Toby (NIST)</i>
	09:45 10:30		<a href="#">using available tools</a> <i>Louis Farrugia (Glasgow)</i>	<a href="#">program and the science of crystallography</a> <i>Tom Spek (Utrecht)</i>	<a href="#">Maximum likelihood and its role in structure solution</a> <i>Kevin Cowtan (York)</i>	<a href="#">structure comparison, analysis &amp; validation</a> <i>Tom Spek (Utrecht)</i>	
	10:30 11:00		coffee				
	11:45		<a href="#">scripting languages</a> <i>Ralf Grosse-Kunstleve (Berkeley)</i>	<a href="#">GUI design</a> <i>Brian Toby (NIST)</i>	<a href="#">Simple algorithms for Macromolecular Refinement</a> <i>George Sheldrick (Göttingen)</i>	<a href="#">refinement I - classical techniques</a> <i>Lynn ten Brink (San Diego)</i>	<a href="#">testing software</a> <i>Harry Powell (Cambridge)</i>
	11:45 12:30	<a href="#">arrival &amp; registration</a>	<a href="#">maintenance &amp; development of legacy code</a> <i>David Watkin (Oxford)</i>	<a href="#">automated data collection and integration</a> <i>Rob Hoft (Bruker-AXS)</i>	<a href="#">automated structure solution</a> <i>Tom Terwilliger (Los Alamos)</i>	<a href="#">refinement II - modern developments</a> <i>Dale Tronrud (Oregon)</i>	<a href="#">Future of Direct Methods</a> <i>George Sheldrick (Göttingen)</i>

lunch		12:30 - 15:00	15:00 - 15:45	15:45 - 16:30	16:30 - 16:45
<a href="#">complete rewrites - when, why and how?</a> <i>Jim Pflugrath (MSC/Rigaku)</i>	<a href="#">operating hardware</a> <i>Rob Hoft (Bruker-AXS)</i>	<a href="#">symmetry in the modern world</a> <i>Kevin Cowtan (York)</i>	<a href="#">Rietveld refinement</a> <i>Juan Rodriguez-Carvajal (CEA-CNRS)</i>	<a href="#">depart for Florence (bus will be provided as part of Siena School registration cost)</a>	
<a href="#">coordinate frames and operators</a> <i>Kevin Cowtan (York)</i>	<a href="#">integration</a> <i>Jim Pflugrath (MSC/Rigaku)</i>	<a href="#">powder crystallography - structure solution</a> <i>Bill David (RAL)</i>	<a href="#">statistical treatment of uncertainties</a> <i>Dale Tronrud (Oregon)</i>		

workshops/projects		16:30 - 17:15	17:15 - 17:30	17:30 - 17:45	17:45 - 18:15	18:15 - 18:45	18:45 - 19:00	19:00 - 19:45	19:45 - 20:00
<a href="#">Welcome from the University of Siena</a> <i>Marcello Mellini</i>	<a href="#">Welcome Davide Viterbo (IUCr Executive Committee representative)</a>	<a href="#">Introductory talk</a> <i>Tom Spek (Utrecht)</i>	<a href="#">Porting between Operating Systems</a> <i>Harry Powell (Cambridge)</i>	<a href="#">"new" algorithms</a> <i>Tom Terwilliger (Los Alamos)</i>	<a href="#">pair distribution functions (PDF)</a> <i>Simon Billinge (Michigan)</i>	<a href="#">3 x student presentations</a>	<a href="#">3 x student presentations</a>	<a href="#">Presentation of the Projects</a> <i>Harry Powell (Cambridge)</i>	<a href="#">dinner (Dinner in downtown Siena on Saturday 20th - included in cost of school registration)</a>

## Thanks to our Sponsors !

- Bruker-Nonius AXS
- Cambridge Crystallographic Data Center
- CCP4
- IUCr
- Max-Inf2
- Merck & Co., Inc, USA
- Oxford Diffraction
- Rigaku/MSC
- Universita degli Studi di Siena

## Porting between Operating Systems

*or*

## how to increase your customer base

Why bother to port (doesn't everyone use MS-Windows)?

- support a wider community (some people use other platforms) - more people will use your software
- future proofing (remember DEC VAX or PDP-11? or IBM 370/168?)
- leads to more standard code (which should be more maintainable)

Harry Powell MRC-LMB

Which platforms/operating systems to port to? This depends on:

- who you want to use your software (most important)
- what hardware you have available for building/testing

Serious developers will usually have access to:

- PC / Linux (no excuse for not having it!)
- PC / MS-Windows (probably)
- Mac / OS X (becoming more popular)
- SGI / Irix (historically important)
- Alpha / Tru64 (historically important)
- Sun /Solaris or SunOS (niche product in crystallography)
- HP-UX (niche product in crystallography)

Three basic types of porting;

1. existing software
2. new graphical interfaces
3. functionality

## Porting existing software

### Porting Existing Software

Assumption that a complete re-write is not necessary/appropriate/desired

Reasons include:

this program is

- mature;
- does its job;
- will not be developed much further;
- enormous (any re-write will take many man-years & introduce many bugs)
- obsolescent (*i.e.* it will be rewritten when time permits)

## Porting Existing Software

Used to be expensive/difficult - only readily available compilers were commercial and tied to a platform, had useful platform dependent features (e.g. VAX FORTRAN)

Now platforms exist which have a “free” development environment (e.g. Linux, Mac OS X, Cygwin/MinGW)

“Free” compilers which adhere to the standards now exist (gcc/g77 for most platforms, Salford FTN77 for MS-Windows, icc/icc for Intel Linux)

## Porting Existing Software

Why use a “free” compiler?

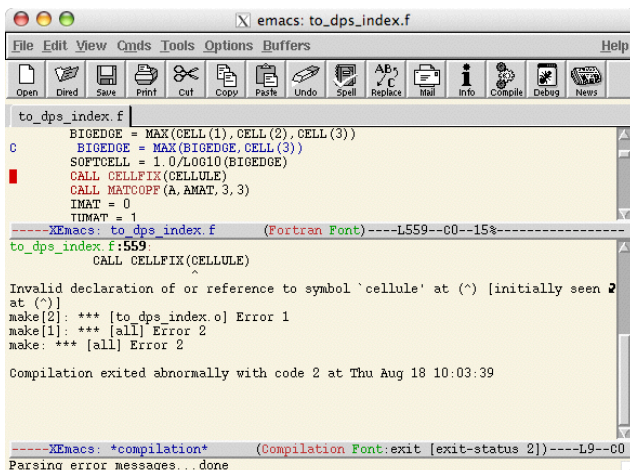
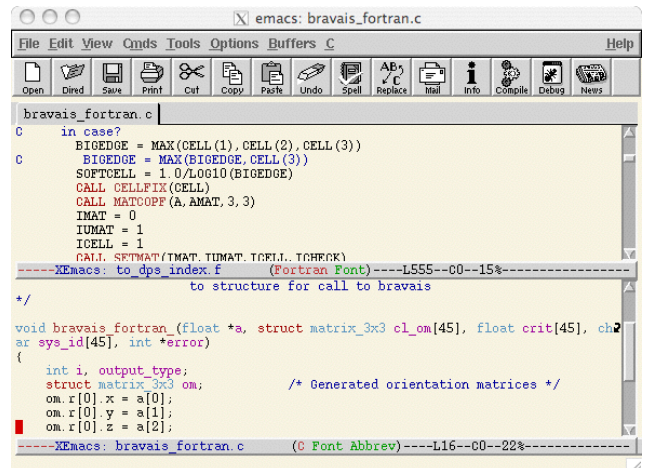
- low initial cost
- modern ones often produce executables about as fast as commercial ones
- same compiler across multiple platforms
- readily available cross compilers (build on Linux, run on Windows - or vice versa)
- good development environments (e.g. XEmacs)

but - you get what you pay for, so it can be buggy - e.g. gcc 2.96 distributed with Red Hat Linux should never have been released

## Porting Existing Software

Why use a commercial compiler?

- “You get what you pay for”
  - user support
  - often better optimization (for specific platforms)
  - good development environments (e.g. Visual Studio)
  - your institution/company may already have paid for and own a licence



## Porting from MS-Windows to other systems

- need a system to build on
- need a system to test on

Possible to cross-develop on MS-Windows but the target windowing system will be different.

Two easy routes -

1. install Linux and dual boot (best environment)
2. install Cygwin (easiest since you can run MS-Windows simultaneously)

Each of these provides a fully-featured windowed environment via X11 windows, but Cygwin will produce MS-Windows executables unless you cross-compile!

#### Porting from any UNIX to other systems

- “easy” if the other system is UNIX-based
- can be an opportunity to modularize functionality
- for MS-Windows need a system (probably not a problem for most people):

##### Choices:

- MinGW (Minimalist GNU for Windows), provides compilation tools under Windows
- Cygwin; easy to migrate a UNIX build (essentially identical to Linux on PC)
- Cross-compilers; build under system A, run on system B.

#### Problems with porting (2):

May identify bugs in compilers - e.g. file “break.f” contains the following:

```
CHARACTER*2 TEST(2)
CHARACTER*1 JUNK
JUNK = 'O'
WRITE(TEST,FMT=1000) JUNK
1000 FORMAT(A)
END
```

```
[g4-15:~/programs] harry% gfortran -c break.f
break.f: In function 'MAIN__':
break.f:3: internal compiler error: Bus error
Please submit a full bug report,
with preprocessed source if appropriate.
See <URL:http://gcc.gnu.org/bugs.html> for instructions.
```

This can lead to a new career in compiler development!

#### Problems with porting (4):

You may feel that months of work may be wasted when a manufacturer makes a major change to its hardware or operating system, e.g.

Apple - OS X made all previous Apple software obsolete

Apple - change of processor from PowerPC to Intel makes xlc/xlf compiler specific work obsolete.

#### Problems with porting (1):

It will probably identify bugs in your code (some “strict” compilers are stricter than others):

```
WRITE(*,*)'hello world'
```

compiles as expected with f77 on Tru64 UNIX, g77 on Linux & MacOS, but not with xlf on MacOS.

```
WRITE(*,*)'hello world';
```

compiles as expected with f77 on Tru64 UNIX, g77 on Linux & MacOS, xlf on MacOS, but not on Irix with f77 (because it's actually f90).

#### Problems with porting (3):

If you can't distribute static binaries, your customer's computers are probably missing vital libraries, shared objects, and other bits and pieces necessary to run your code which you

- (a) didn't think were necessary
- (b) assumed were ubiquitous
- (c) hadn't even thought about

They might be there, but are from an older/newer version of the operating system and are therefore incompatible.

#### Problems with porting (5):

Porting to (and from) MS-Windows is probably the most difficult step because of its unique environment; all other popular platforms share a similar operating system and have available a similar windowing system.

For example, *identical* code for Mosflm\* (150,000 lines of Fortran, 100,000 lines of C, + 80,000 lines of graphics library) builds using the same commands on all UNIX boxes, but without tools like Cygwin requires a major restructuring for MS-Windows.

\* a popular data integration program

## new graphical interfaces

Command-line interfaces - old-fashioned but generally straightforward; however, MS-Windows users won't like it!

Graphical interfaces - need to distinguish between

- windowing environment - most platform dependent
  - MS-Windows - tied to Windows PCs
  - Aqua - tied to Macintosh
  - X11 - can be used on anything it's been ported to
- interface utilities -
  - Tk (Tcl/Tk, Tkinter, ...)
  - wxWidgets (*was* wxWindows), Qt, Clearwin...
  - browser applets

For portability we need the utility to have been ported to the environment (or the environment to have been ported to the platform)

The current Mosflm approach to a GUI -

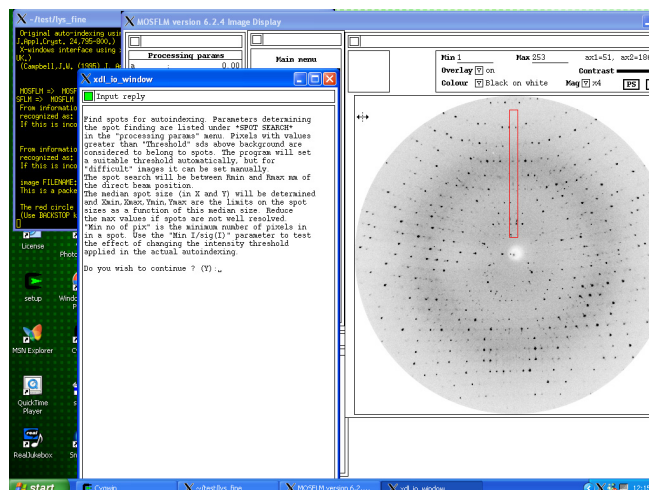
- uses a custom-built (obsolete) X11 widget set - `xdl_view`
- compile to create a monolithic program

Pros:

- distribution of the executable is easy (no need for extra libraries for the GUI)
- familiar widget set (for the programmer, anyway)

Cons:

- development of both Mosflm and the GUI are intimately linked
- "hard" to port to OS's without an X-server
- `xdl_view` can behave subtly differently on different OS's - relied on some benign features of compilers



The new Mosflm approach to a GUI;

- core crystallographic functions (controlled by command line instructions)
- A Tcl/Tk GUI which reads XML and writes native Mosflm commands
- the two parts communicate *via* TCP/IP sockets

Pros:

- Mosflm can be run on any platform independently of the GUI
- development of both Mosflm and the GUI can proceed semi-independently

### Portable interface utilities 1 - Tcl/Tk

mature but losing popularity; development in some important modules is almost non-existent (*e.g.* `incrTcl`).

Installation of Tcl/Tk based programs may require additional packages to be installed (*e.g.* `TkImag`), but some are not available for all platforms (*e.g.* BLT is not available for Aqua).

Tkinter is Python's *de facto* standard GUI package. It is a thin object-oriented layer on top of Tcl/Tk.

can interface with compiled code via 4 basic routes -

- embed compiled code into Tcl/Tk script
- run external programs directly from the script
- read from & write to a file
- read from & write to a TCP/IP socket

## Portable interface utilities 2 - wxWidgets

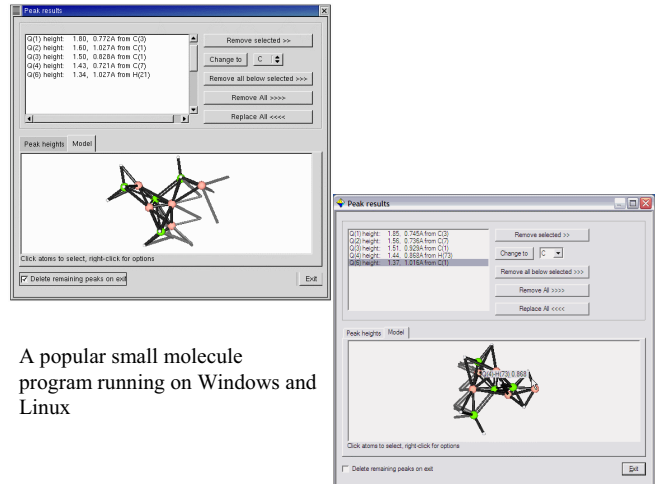
Set of C++ class libraries

Main differences from Tcl/Tk:

- encourages an object oriented approach
- the GUI can be compiled so that the programming details can be hidden from the user; also this can give a performance advantage, especially for interactive graphics.

Gives a native look and feel on different platforms, usable with any common C++

wxPython is “a blending of wxWidgets with Python” - used in PHENIX



A popular small molecule program running on Windows and Linux

Could be viewed as a new project - coding from scratch - probably best to use new languages (C++, Python, Java) rather than established languages.

What you use depends on the application to some extent, e.g.  
web applications - Java  
heavyweight applications - C++  
prototyping - Python

If you have a free choice at this point, development can be faster and portable features can be designed in.

## Porting functionality

Why not just use FORTRAN?

- fast executables with little effort
- many applications/libraries available
- well-established amongst senior scientists
- F90, 95, 200x have modern programming constructs

Why *shouldn't* you use FORTRAN?

- can quickly lead to spaghetti code
- possible lack of long-term maintainability
- little support from “real” programmers (it isn't taught in CS departments!)
- modern languages make development faster
- good OOP imposes modularity

Don't re-invent the wheel unnecessarily - there is a whole host of functionality which has already been coded (software libraries for underlying crystallographic operations) - but there may be licensing issues.

Code can be cribbed from books (e.g. "*Numerical Recipes*") but be aware of copyright issues.

Other developers may be willing to "give" you their code for inclusion (e.g. both autoindexing routines in Mosflm)

Finally -

Design in portability -

- don't use platform specific features if possible
- use standard programming constructs
- gcc/g77/g++ is available for virtually every platform - essentially the same build can be used (some compiler flags and libraries may differ)



## Modern approaches to programming

Ralf W. Grosse-Kunstleve

Computational Crystallography Initiative  
Lawrence Berkeley National Laboratory

## Disclosure



- **Experience**
  - Basic
  - 6502 machine language
  - Pascal
  - Fortran 77
  - csh, sh
  - C
  - Perl
  - Python
  - C++
- **Last five years**
  - Python & C++ -> cctbx, phenix
- **Development focus**
  - phenix.refine, phenix.hyss
- **No experience**
  - TCL/TK
  - Java

## Computational Crystallography Toolbox



- **Open-source component of phenix**
  - Automation of macromolecular crystallography
- **mmtbx** – macromolecular toolbox
- **cctbx** – general crystallography
- **scitbx** – general scientific computing
- **libtbx** – self-contained cross-platform build system
- **SCons** – make replacement
- **Python** scripting layer (written in C)
- **Boost** C++ libraries
- **Exactly two external dependencies:**
  - OS & C/C++ compiler

## Object-oriented programming



The whole is more than the sum of its parts.

Syntax is secondary.

## Purpose of modern concepts



- **Consider**
  - You could write everything yourself
  - You could write everything in machine language
- **Design of Modern Languages**
  - Support large-scale projects <-> Support collaboration
  - Maximize code reuse <-> Minimize redundancy
  - Software miracle: improves the more it is shared

## Main concepts behind modern languages



- **Namespaces**
- **A special namespace: class**
- **Polymorphism**
- **Automatic memory management**
- **Exception handling**
- **Concurrent development**
  - Developer communication
- **Secondary details**
  - friend, public, protected, private



## Evolution of programming languages



### Namespaces

- Emulation

```
MtzSomething (CCP4 CMTZ library)
http://www.ccp4.ac.uk/dist/html/C\_library/cmtzlib\_8h.html
QSomething (Qt GUI toolkit)
http://doc.trolltech.com/4.0/classes.html
PySomething (Python)
http://docs.python.org/api/genindex.html
glSomething (OpenGL library)
http://www.rush3d.com/reference/cgOpenGL-bluebook-1.0/
A00, A01, C02, C05, C06 (NAG library)
http://www.nag.co.uk/numeric/r/manual/html/FLlibrarymanual.asp
```

- Advantages

- Does not require support from the language

- Disadvantages

- Have to write XXXSomething all the time
- Nesting is impractical

## Evolution of programming languages



### Namespaces

- Formalization

similar to:

transition from flat file systems to files and directories

```
namespace MTZ {
    Something
}
```

- Disadvantages

- Does require support from the language

- Advantages

- Inside a namespace it is sufficient to write Something
  - as opposed to XXXSomething
- Nesting “just works”
  - If you know how to work with a directories you know how to work with namespaces

## Evolution of programming languages



### A special namespace: class

- Emulation

- COMMON block with associated functions

```
double precision a, b, c, alpha, beta, gamma
COMMON /unit_cell/ a, b, c, alpha, beta, gamma
subroutine ucinit(a, b, c, alpha, beta, gamma)
double precision function ucvol()
double precision function stol(h, k, l)
```

- Disadvantage

- The associations are implicit
  - difficult for others to see the connections

## Evolution of programming languages



### A special namespace: class

- Formalization

```
class unit_cell:
    def __init__(self, a, b, c, alpha, beta, gamma)
    def vol(self)
    def stol(self, h, k, l)
```

- What's in the name?

- class, struct, type, user-defined type

- Advantage

- The associations are explicit
  - easier for others to see the connections

## Evolution of programming languages



### A special namespace: class

- Formalization

```
class unit_cell:
    def __init__(self, a, b, c, alpha, beta, gamma)
    def vol(self)
    def stol(self, h, k, l)
```

- What's in the name?

- class, struct, type, user-defined type

- Advantage

- The associations are explicit
  - easier for others to see the connections

## Evolution of programming languages



### A namespace with life-time: self, this

- COMMON block = only one instance
- class = blueprint for creating arbitrarily many instances

- Example

```
hex = unit_cell(10, 10, 15, 90, 90, 120)
rho = unit_cell(7.64, 7.64, 7.64, 81.79, 81.79, 81.79)
```

- hex is one instance, rho another of the same class
- Inside the class definition hex and rho are both called self

- What's in the name?

- self, this, instance, object

- hex and rho live at the same time

- the memory for hex and rho is allocated when the object is constructed

## Life time: a true story



A true story about my cars, told in the Python language:

```
class car:
    def __init__(self, name, color, year):
        self.name = name
        self.color = color
        self.year = year

car1 = car(name="Toby", color="gold", year=1988)
car2 = car(name="Emma", color="blue", year=1986)
car3 = car(name="Jamson", color="gray", year=1990)
del car1 # donated to charity
del car2 # it was stolen!
car4 = car(name="Jessica", color="red", year=1995)
```

## Alternative view of **class**



- Function returning only **one** value

```
real function stol(x)
...
s = stol(x)
```

- Function returning **multiple** values

```
class wilson_scaling:
    def __init__(self, f_obs):
        self.k = ...
        self.b = ...
wilson = wilson_scaling(f_obs)
print wilson.k
print wilson.b
```

- Class is a generalization of a function

## Evolution of programming languages



A special namespace: **class**

- Summary
  - A class is a namespace
  - A class is a blueprint for object **construction** and **deletion**
  - In the blueprint the object is called **self** or **this**
  - Outside the object is just another variable
- When to use classes?
  - Only for “big things”?
  - Is it expensive?
- Advice
  - If you think about a group of data as one entity
    - use a class to formalize the grouping
  - If you have an algorithm with 2 or more result values
    - implement as class

## Evolution of programming languages



### Polymorphism

- The same source code works for different types
- **Runtime** polymorphism
  - “Default” in dynamically typed languages (scripting languages)
  - Very complex in statically typed languages (C++)
- **Compile-time** polymorphism
  - C++ templates

## Evolution of programming languages



### Compile-time polymorphism

- Emulation
  - General idea

```
S  subroutine seigensystem(matrix, values, vectors)
D  subroutine deigensystem(matrix, values, vectors)
S  real      matrix(...)
D  double precision matrix(...)
S  real      values(...)
D  double precision values(...)
S  real      vectors(...)
D  double precision vectors(...)
```

Use **grep** or some other command to generate the single and double precision versions
  - Real example
    - <http://www.netlib.org/lapack/individualroutines.html>

## Evolution of programming languages



### Compile-time polymorphism

- Formalization

```
template <typename FloatType>
class eigensystem
{
    eigensystem(FloatType* matrix)
    { // ...
    }
};

eigensystem<float> es(matrix);
eigensystem<double> es(matrix);
```
- The C++ template machinery **automatically** generates the type-specific code **as needed**

## Automatic memory management



- Context

- Fortran: **no** dynamic memory management
  - Common symptom
    - Please increase MAXA and recompile
- C: **manual** dynamic memory management via malloc & free
  - Common symptoms
    - Memory leaks
    - Segmentation faults
    - Buffer overruns (vector for virus attacks)
    - Industry for debugging tools (e.g. purify)

## Automatic memory management



- Emulation: Axel Brunger's ingenious approach

- Insight: stack does automatic memory management!

```
subroutine action(args)
  allocate resources
  call action2(args, resources)
  deallocate resources

subroutine action2(args, resources)
  do work
```

- Disadvantage
  - Cumbersome (boiler plate)

## Automatic memory management



- Formalization

- Combination
  - Formalization of object construction and deletion (class)
  - Polymorphism
- Result = fully automatic memory management
- "Default" in scripting languages
  - garbage collection, reference counting
- C++ Standard Template Library (STL) container types
  - std::vector<T>
  - std::set<T>
  - std::list<T>

- Advice

- Use the STL container types
- **Never** use new and delete
  - Except in combination with smart pointers
    - std::auto\_ptr<T>, boost::shared\_ptr<T>

## Evolution of programming languages



### Exception handling

- Emulation

```
subroutine matrix_inversion(a, ierr)
  ...
  matrix_inversion(a, ierr)
  if (ierr .ne. 0) stop 'matrix not invertible'
```

- Disadvantage

- **ierr** has to be propagated and checked throughout the call hierarchy -> serious clutter
- to side-step the clutter: **stop**
  - not suitable as library

## Emulation of exception handling



```
program top
  call high_level(args, ierr)
  if (ierr .ne. 0) then
    write(6, *) 'there was an error', ierr
  endif
end

subroutine high_level(args, ierr)
  call medium_level(args, ierr)
  if (ierr .ne. 0) return
  do something useful
end

subroutine medium_level(args, ierr)
  call low_level(args, ierr)
  if (ierr .ne. 0) return
  do something useful
end

subroutine low_level(args, ierr)
  if (args are not good) then
    ierr = 1
    return
  endif
  do something useful
end
```

## Evolution of programming languages

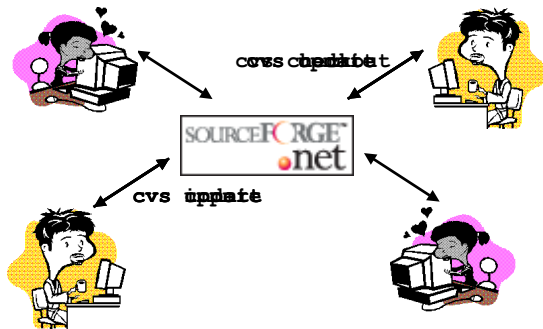


### Exception handling

- Formalization

```
def top():
  try:
    high_level(args)
  except RuntimeError, details:
    print details
def high_level(args):
  medium_level(args)
  # do something useful
def medium_level(args):
  low_level(args)
  # do something useful
def low_level(args):
  if (args are not good):
    raise RuntimeError("useful error message")
  # do something useful
```

## Collaboration via SourceForge



## Conclusion concepts



- **Advantages**
  - Modern languages are the result of an evolution
    - Superset of more traditional languages
    - A real programmer can write Fortran in any language
  - Designed to support large collaborative development
    - However, once the concepts are familiar even small projects are easier
  - Solve common problems of the past
    - memory leaks
    - error propagation from deep call hierarchies
  - Designed to reduce redundancy (boiler plate)
  - If the modern facilities are used carefully the boundary between "code" and documentation begins to blur
    - Especially if runtime introspection is used as a learning tool
  - Readily available and mature
    - C and C++ compilers are at least as accessible as Fortran compilers
  - Rapidly growing body of object-oriented libraries

## Conclusion concepts



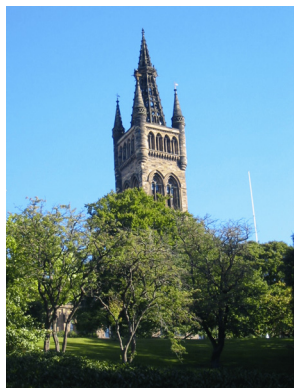
- **Disadvantages**
  - It can be difficult to predict runtime behavior
    - Tempting to use high-level constructs as black boxes
  - You have to absorb the concepts
    - syntax is secondary!
  - However: Python is a fantastic learning tool that embodies all concepts outlined in this talk
    - except for compile-time polymorphism

## Acknowledgements



- **Organizers of this meeting**
- Paul Adams
- Pavel Afonine
- Peter Zwart
- Nick Sauter
- Nigel Moriarty
- Erik McKee
- Kevin Cowtan
- David Abrahams
- Open source community

<http://www.phenix-online.org/> <http://cctbx.sourceforge.net/>



**Louis Farrugia**

## **Lecture-1**

### **Using available tools**

1. Types of software tools available
2. How to use them in your software ?
3. Where to get them from ?

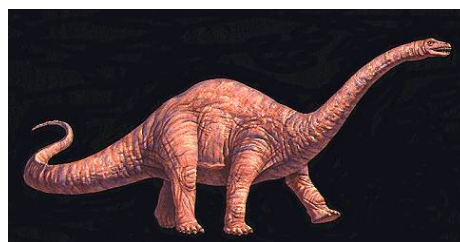
Louis Farrugia - IUCr Computing School - Siena 2005 - Using Available Tools

## **From the Computing School website ...**

### **School History**

First the earth cooled, then there were dinosaurs..

Louis Farrugia - IUCr Computing School - Siena 2005 - Using Available Tools



### **FORTRANOSAURUS – a real dinosaur ?**

1. FORTRAN is an ancient language from 1950's – extinction long predicted, but still being maintained and modernised – FORTRAN95 in current use. FORTRAN 2003 is latest standard (but as yet no compilers !)
2. FORTRAN is an efficient language and well suited to scientific computing
3. There exist vast libraries of FORTRAN sources for scientific computing
4. It is *possible* to write modern software in FORTRAN.

Louis Farrugia - IUCr Computing School - Siena 2005 - Using Available Tools

### **Types of software tools currently available**

- programming languages & compilers/interpreters
- GUI (graphical user interfaces)
- 'scientific' libraries
- pre-built 'scientific' applications
- utility applications

Louis Farrugia - IUCr Computing School - Siena 2005 - Using Available Tools

### **Programming languages used in crystallographic programs and libraries**

- FORTRAN
- C
- C++
- Java, Python
- other scripting & GUI languages

Modern applications may be multi-language

## GUI's – Graphical User Interfaces

GUI's are now ubiquitous and seemingly necessary

**Advantages of the GUI approach**

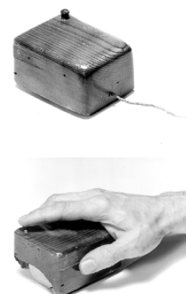
- easy to use and learn programs - commonality of interface
- scientific programs - programmer regulates numerical input
- options/possible pathways made obvious - less need for extensive reference manuals

**Disadvantages of the GUI approach**

- too easy to use and learn programs - no understanding of underlying methodologies
- scientific programs - too restrictive for complex problems
- reluctance to read extensive reference manuals

## GUI's – Graphical User Interfaces

Makes full use of VDU screen with a pointing device - mouse (patented 1964)



## GUI's – Graphical User Interfaces

**Disadvantages of the GUI approach**

- too easy to use and learn programs

I have question about WinGX software.

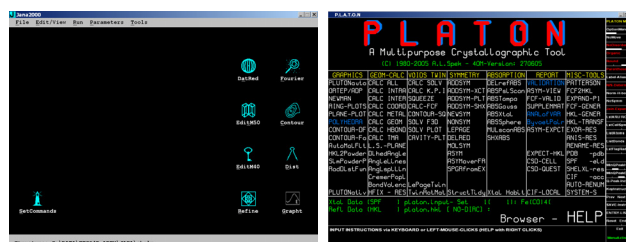
when I launched DATA -- > Cad4 -- > XCad4 , the program displays :

```
5464 reflections processed, of which 56 were standards (5408 netto)
593 reflection(s) with zero or negative intensity
0 reflection(s) rejected with intensity less than -9999.00
1 reflection(s) deviated from scan centre by more than DAMG
0 reflection(s) not measured because collision predicted
0 reflection(s) not measured because chie > 100 deg
999 reflection(s) measured on pre-scan only
0 reflection(s) with a count loss > 1%
0 reflection(s) with too strong intensities
0 reflection(s) with bad backgrounds
```

My question is : what are standards ?

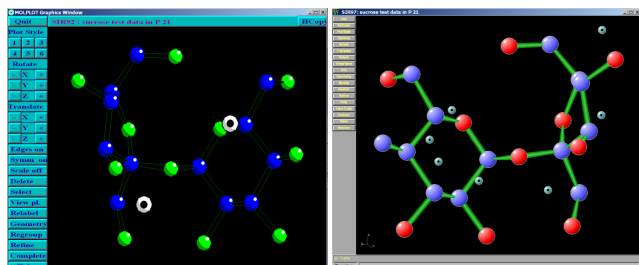
## Ways of producing GUI's for applications

- Do it yourself
- Use GUI libraries



JANA2000

PLATON



SIRWARE Programs SIR92 / SIR97  
<http://www.ic.cnr.it/>

GUI libraries – many commercial but some free. Many are cross-platform. A small sample ...

- Java
- Python - <http://python.org/>
- wxWidgets - <http://www.wxwidgets.org/>
- wxPython - <http://www.wxpython.org/>
- Tcl/tk - <http://tcl.activestate.com/>
- Qt - <http://www.trolltech.com/tip>
- VGUI - <http://vgui.sourceforge.net/>
- GTK+ (Gimp tool kit)- <http://www.gtk.org/>
- Fast Light Toolkit - <http://www.fltk.org/>
- FOX - <http://www.fox-toolkit.org/>

Many are written in C/C++, so interfacing to languages like FORTRAN is difficult - Steep learning curve to use.

“Scientific” libraries – mathematical and crystallographic  
Some examples ...

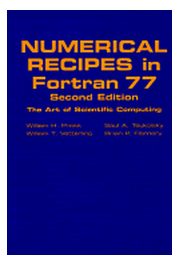
- **NAG** – Numerical Algorithms Group
- **CCSL** – Cambridge Crystallographic Subroutine Library
- **CCTBX** – Computational Crystallographic Toolbox – Ralf Grosse-Kunstleve
- **CLIPPER** – OO crystallographic libraries – Kevin Cowtan
- **CrysFML** – Crystallographic Fortran 95 Modules Library - Juan Rodriguez- Carvajal – LLB (Fullprof suite)
- **CIFTBX** – tools for reading/writing CIF's (also CIFLIB for mmCIF)
- **GETSPEC** – tools for space group symmetry
- **FPRIME** – tool for X-ray dispersion corrections

- crystallography source code archives & “museums”
- graphics libraries

Numerical algorithms  
group  
<http://www.nag.co.uk>

languages  
**FORTAN & C**

BLAS - <http://www.netlib.org/blas/>  
LAPACK - <http://www.netlib.org/lapack/>



William H. Press  
Brian P. Flannery  
Saul A. Teukolsky  
William T. Vetterling

languages

FORTAN 77/90/95

C, C++

<http://www.numerical-recipes.com>

## Crystallographic library tools

Jane Brown - <http://www.ill.fr/dif/ccsl/html/ccsldoc.html>  
language - **FORTAN**

**FORTAN API - Designed to be incorporated into a program**

**memory containing CIF is shared between library and application**

**later versions offer XML output**

<http://www.iucr.org/iucr-top/index.html>



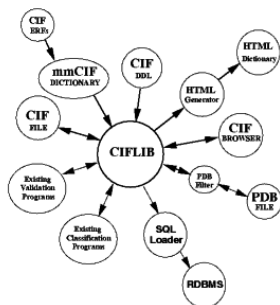
- |   |   |
|---|---|
| <pre> program main include &lt;cfirun.cmn&gt; logical      fl character*10 buf1,buf2,buf3,buf4,buf5,buf6,buf7  call getarg(1,buf4) if(buf4 == '1') stop fl = init(10,11,12,5) fl = occl('bypass.cir') if(.not. fl) stop 'CIF not found' fl = data('bond_valence_parameters') if(.not. fl) stop 'Block not found' n = 0 do   fl = char('valence_param_atom_1',buf1)   if(.not. loop) exit   if(buf1 /= buf4) cycle   fl = char('valence_param_atom_2',buf2)   if(buf2 /= buf4) cycle   fl = char('valence_param_atom_valence',buf3)   fl = char('valence_param_atom_valence',buf3)   fl = char('valence_param_atom_2_valence',buf4)   fl = char('valence_param_atom_2_valence',buf4)   fl = char('valence_param_atom_val',buf5)   if(buf5 == '9') buf5 = '?'   if(buf4 == '9') buf4 = '?'   n = n + 1   write('(*)') buf1,buf3,buf2,buf4,buf5,buf6 enddo write('(*)') n end </pre> | <pre> program main include &lt;cfirun.h&gt; logical      fl character*10 buf1,buf2,buf3,buf4,buf5,buf6,buf7  call getarg(1,buf4) if(buf4 == '1') stop call cfirun_init() fl = init(10,11,12,5) fl = occl('bypass.cir') if(.not. fl) stop 'CIF not found' fl = data('bond_valence_parameters') if(.not. fl) stop 'Block not found' n = 0 do   fl = char('valence_param_atom_1',buf1)   if(.not. loop) exit   if(buf1 /= buf4) cycle   fl = char('valence_param_atom_2',buf2)   if(buf2 /= buf4) cycle   fl = char('valence_param_atom_1_valence',buf3)   fl = char('valence_param_atom_2_valence',buf4)   fl = char('valence_param_atom_val',buf5)   fl = char('valence_param_atom_val',buf5)   if(buf4 == '9') buf4 = '?'   if(buf5 == '9') buf5 = '?'   n = n + 1   write('(*)') buf1,buf2,buf3,buf4,buf5,buf6 enddo write('(*)') n end </pre> |
|---|---|

12,288 bytes

- advantages – easy to read CIF and write out structured CIF's
- disadvantages – will not read a CIF with syntax error(s)  
only loads one CIF

## GETSPEC

**James Hester – ANU implemented in Python**  
<http://www.ansto.gov.au/natfac/ANBF/CIF/>



**C Class Library – Rutgers**  
***J. Appl. Cryst.* (1997) 30, 79-83.**

designed for mmCIF

- written by I David Brown, McMaster
- incorporated into WinGX
- incorporated into LMPG software of Jean Laugier
- FORTRAN source code deposited at CCP14

**This program calculates the symmetry operators (general positions) and special positions for any setting of any space group based on the Hall space group symbol**

**SGINFO (C code) – Ralf Grosse-Kunstleve → CCTBX**

**GETSPEC**

**Space Group Setting P213**

Space group symbol: **P213** Browse...

Enter the short **Hermann Mauguin** symbol as given in International Tables Volume A  
 P 2<sub>1</sub> 3  
 P 2<sub>1</sub> 3

Space group data  
 Space group: **P213**  
 Setting: standard  
 Equivalent positions: 12  
 Space group multiplicity: 12

Hall symbol: **P 2<sub>1</sub> 3**  
 Unit cell axes: **a=1, b=1, c=1**  
 Lattice type: **P-primitive**  
 Point group: **23**  
 Lattice points: **1**  
 Centrosymmetric: **no**  
 Extra information: **chiral**

Special positions (xyz, multiplicity, point symmetry):

x, y, z	12	1
x, y, x	4	3, 2

Go LATT/SYMM Cancel

Read Hermann-Mauguin symbol

```
integer function gspHall(Hall_symbol)
interprets Hall symbol and loads Seitz matrices in memory
```

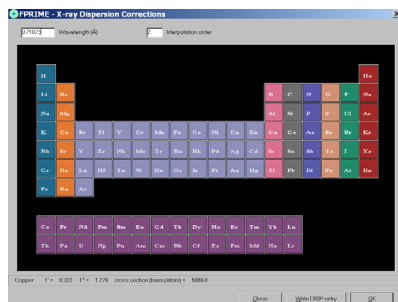
*integer function gspNsym()*  
returns number of independent symmetry operations

*integer function gspNLatt()*  
returns number of lattice translations

```
subroutine gspSymXYZ(n,string)
returns xyz notation for symmetry operations into character array
string(n)
```



## FPRIME



Code by Bob von Dreele  
Los Alamos N'tnl Lab

Also by Sean Brennan

Both implemented in  
WinGX

Calculates  $f'$  and  $f''$  and  
abs cross-section

FORTTRAN sources

<ftp://ftp.lanl.gov/public/gsas/windows>  
[ftp://apollo.apsl.anl.gov/pub/cross-section\\_codes/](ftp://apollo.apsl.anl.gov/pub/cross-section_codes/)

<http://www-phys.llnl.gov/Research/scattering/asf.html>

## Anomalous Scattering Factors

Jump to:  
[More Information](#) | [return home](#)

### ASF Calculation

Atomic symbol or number (H-Es, or 1-99)

☒ Single energy

photon energy (keV, 0-10000)

☐ Multiple energies

minimum energy (keV, min. 0)

maximum energy (keV, max. 10000)

energy increment (0 => default grid)

## Crystallographic Webservers

**Bilbao Crystallographic  
Server**

<http://www.cryst.ehu.es/>

[A Web Site with  
Crystallographic Tools  
Using the International  
Tables for  
Crystallography]

Space Groups Retrieval Tools	
GENPOS	Generators and General Positions of Space Groups
WYCKPOS	Wyckoff Positions of Space Groups
WYCKSETS	Equivalent Sets of Wyckoff Positions
NORMALIZER	Normalizers of Space Groups
KVEC	The k-vector types of Space Groups
Group - Subgroup Relations of Space Groups	
MAXSUB	Maximal Subgroups of Space Groups
SERIES	Series of Maximal Isomorphic Subgroups of Space Groups
SUBGROUPGRAPH	Lattice of Maximal Subgroups
CELLSUB	List of subgroups for a given k-index
HERMANN	More group-subgroup relations
COMMONSUBS	Common Subgroups of Two Space Groups
TRANSPATH	Transition Paths
MINSUP	Minimal Supergroups of Space Groups
SUPERGROUPS	Supergroups of Space Groups
WYCKSPLIT	The splitting of the Wyckoff Positions
Representation Theory Applications	
REPRES	Space Groups Representations
COREL	Correlations Between Representations
POINT	Point Group Tables

## Crystallography source code archives & "museums"

• <http://www.ccp14.ac.uk/> - mirror sites for much software

• <http://sdpd.univ-lemans.fr/museum/> - Armel le Bail

• <http://www.chem.gla.ac.uk/~paul/GX/> - GX source code

• <http://www.ccp14.ac.uk/ccp/ccp14/ftp-mirror/larryfinger>

## Application tools

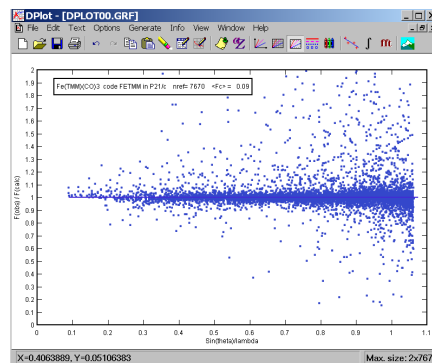
Data plotting- DPLOTT - <http://www.dplot.com>



Dplot is  
commercial  
software –  
scientific data  
plotting

a large number  
of formats

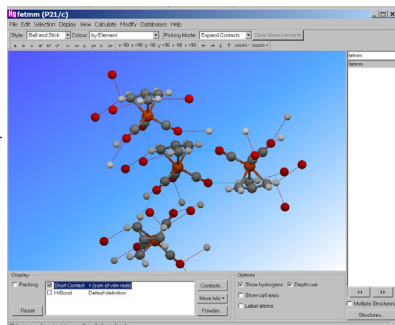
very flexible  
presentation



CCDC programs - <http://www.ccdc.cam.ac.uk/>

#### Mercury features include:

- Input of hit-lists from ConQuest, or other format files such as CIF, PDB, MOL2 and MOLfile
- Location and display of intermolecular and/or intramolecular hydrogen bonds, short nonbonded contacts, and user-specified types of contacts
- The ability to build and visualise a network of intermolecular contacts



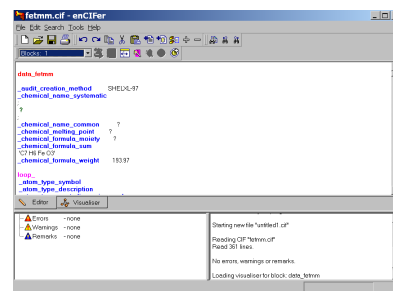
CCDC programs - <http://www.ccdc.cam.ac.uk/>

#### enCIFer - CIF checking, editing and visualisation software from the CCDC

CIF (Crystallographic Information File) is now a standard means of information exchange in crystallography.

It is also the recommended way of submitting data to the CSD.

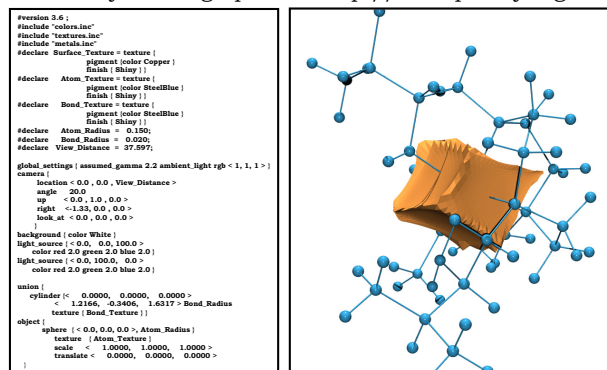
enCIFer provides an intuitive, user-friendly interface to add information safely to the resultant CIF without corrupting the strict syntax.



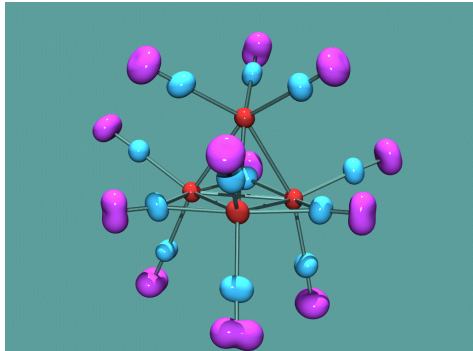
Ray traced graphics - <http://www.povray.org>



Ray traced graphics - <http://www.povray.org>



Ray traced graphics - <http://www.povray.org>



The adp's of  $\text{Rh}_4(\text{CO})_{12}$  shown as the RMSD surface (PEANUT plot)

#### Debugging tools

## Static code analysers

•FTNCHEK <http://www.dsm.fordham.edu/~ftnchek/> for FORTRAN code

•SPLINT <http://www.splint.org/> for C & C++ code

**ftnchek** is a static analyzer for Fortran 77 programs. Its purpose is to assist the user in finding semantic errors. Semantic errors are legal in the Fortran language but are wasteful or may cause incorrect operation.

**Splint** is a tool for statically checking C programs for security vulnerabilities and coding mistakes.

SPAG – plusFORT – <http://www.polyhedron.co.uk>

Converts legacy FORTRAN-66 code into F77 or F95 compliant code – cures the “rats-nest” problem.

### before

```
148 IF(VIEW)152,150,152
150 W(12,1)=1.
    W(12,2)=1.
    IF(W(6,1)-W(6,2))165,175,175
152 DO 160 I=1,2
    DO 155 J=10,12
155 W(J,I)=-W(J-6,I)
    W(12,I)=W(12,I)+VIEW
160 W(13,I)=VV(W(10,I),W(10,I))
    IF(W(13,2)-W(13,1))165,175,175
```

### after

```
100 if ( view/=0 ) then
    do i = 1,2
        do j = 10,12
            w(j,i) = -w(j-6,i)
        enddo
        w(12,i) = w(12,i) + view
        w(13,i) = vv(w(10,i),w(10,i))
    enddo
    if ( w(13,2)>=w(13,1) ) goto 200
else
    w(12,1) = 1.
    w(12,2) = 1.
    if ( w(6,1)>=w(6,2) ) goto 200
endif
```

## Where to get software tools ?

- Web search engines
- CCP14 - <http://www.ccp14.ac.uk/> a web-site with vast number of downloads and links
- SINCRIS - <http://www-ext.lmcp.jussieu.fr/sincris-top/logiciel/>



## Spectrum of languages

Ralf W. Grosse-Kunstleve

Computational Crystallography Initiative  
Lawrence Berkeley National Laboratory

## Spectrum of implementation languages



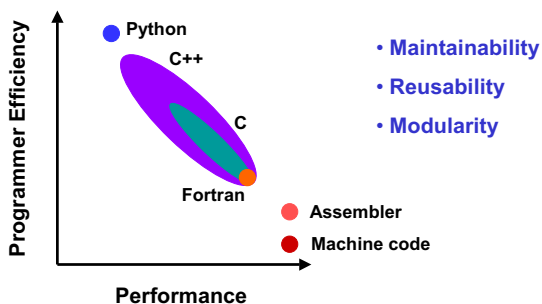
- **Python**
  - Interpreted, Object Oriented, Exception handling
- **C++**
  - Compiled, Object Oriented, Exception handling
- **C**
  - Compiled, User defined data types, Dynamic memory management
- **Fortran**
  - Compiled, Some high-level data types (N-dim arrays, complex numbers)
- **Assembler**
  - Computer program is needed to translate to machine code
- **Machine code**
  - Directly executed by the CPU

## Matrix of language properties



	Dynamically typed -> convenience	Statically typed -> speed
Interpreted -> convenience	Python	Java
Compiled to machine code -> speed	Psyco	C++

## Programmer Efficiency & Performance



## Choice of implementation languages



- **Python**
  - + Very high-level programming
  - + Easy to use (**dynamic typing**)
  - + Fast development cycle (no compilation required)
  - Too slow for certain tasks
- **C++**
  - + High-level or medium-level programming
  - Many arcane details (**strong static typing**)
  - + Largely automatic dynamic memory management (templates)
  - + Much faster than Python
  - + With enough attention, performance even rivals that of FORTRAN

## Happy marriage: Python and C++



- Syntactic differences put aside, Python and C++ objects and functions are very **similar**.
- Flexibility (interpreted, dynamically typed) and Efficiency (compiled, statically typed) are **complementary**.
- Boost.Python (C++ library) provides the link:
  - Non-intrusive on the C++ design
  - Pseudo-automatic wrapping using C++ template techniques
  - No external tools needed
  - Creates sub-classable Python types
  - Python bindings are very maintainable
  - Tutorial and reference documentation

```
class _unit_cell("unit_cell")
    .def("volume", &unit_cell::volume)
    .def("fractionalize", &unit_cell::fractionalize)
;
```

## Vector operations



- **Computer Science wisdom:**
  - Typically 90% of the time is spent in 10% of the code
- **Similar to idea behind vector computers:**
  - Python = Scalar Unit
  - C++ = Vector Unit
- **Loading the vector unit: (8.7 seconds)**

```
miller_indices = flex.miller_index()
for h in xrange(100):
    for k in xrange(100):
        for l in xrange(100):
            miller_indices.append((h,k,l))
```

- **Go! (0.65 seconds)**

```
space_group = sgtbx.space_group_info("P 41 21 2").group()
epsilons = space_group.epsilon(miller_indices)
```

Computing 1 million epsilons takes only 0.65 seconds!

## Compiled vs. Interpreted



- **Compiler**
  - generates fast machine code
- **Interpreter (Python, Perl, TCL/TK, Java)**
  - may generate byte-code but not machine code

## Compiled vs. Interpreted



- **Compiler**
  - generates fast machine code
  - needs arcane compilation commands
  - needs arcane link commands
  - generates object files (where?)
  - may generate a template repository (where?)
  - generates libraries (where?)
  - generates executables (where?)
  - needs a build tool (make, SCons)
  - all this is platform dependent
- **Interpreter (Python, Perl, TCL/TK, Java)**
  - may generate byte-code but not machine code

## Conclusion languages



- **It is important to know the modern concepts**
  - Especially for ambitious projects
- **Syntax is secondary**
  - Anything that does the job is acceptable
    - Python, C++, csh, sh, bat, Perl, Java
- **There is no one size fits all solution**
  - But Python & C++ covers the entire spectrum
- **Carefully weigh programmer efficiency vs. runtime efficiency**
  - Prefer a scripting language unless runtime efficiency is essential

## Acknowledgements



- Organizers of this meeting
- Paul Adams
- Pavel Afonine
- Peter Zwart
- Nick Sauter
- Nigel Moriarty
- Erik McKee
- Kevin Cowtan
- David Abrahams
- Open source community

<http://www.phenix-online.org/> <http://cctbx.sourceforge.net/>

# Legacy Codes

## Do They Have Any Value?

David Watkin, Chemical Crystallography Laboratory  
OXFORD

Siena, 2005

Legacy code can loosely be defined as any code written at 'some time in the past', and can be divided into two categories – *Obsolescent* and *Obsolete* code.

*Obsolescent* code includes programs which clearly have a limited future dynamic life span and will include almost all programs in current use in small molecule crystallography. Their development and maintenance is endangered because they are the product either of a single author, or of a small localised group of authors. In addition, most current small molecule programs are written in FORTRAN, a language which itself will either die out, or transform beyond recognition.

*Obsolete* code includes program which are no longer in active use, and includes forerunners of current programs, programs which are unsupported, and programs addressing issues which have gone away.

Because of the decline in do-it-yourself programming in most crystallography laboratories, legacy code now interests a very limited audience. Outside of the LINUX community, a decreasing number of crystallographers are able to compile old codes. This means that legacy code really only has a value to those small groups of crystallographers still actively developing new programs.

Potential uses for old codes include:

1. Ability to perform computations not in current active codes. Acta Cryst regularly publishes notes or full papers about programs which contain algorithms useful for special purposes, yet which fail to gain common use. If the original author can be contacted, it is sometimes possible to obtain copies of the code. If it is available and can be compiled, the problem is solved.
2. If part of a computation being carried out by a new program is available in an old program, it may be possible to use the old program to validate the new one.
3. As a source of information at a more detailed level than that found in published papers. An example of this is the ability to model electron density distributed throughout some shape other than the usual sphere or ellipsoid. In 1950 King and Lipscomb reported the mathematics for some simple shapes. In 1975 Bennett, Hutcheon and Foxman reported use of a program they had written to carry out these computations, and similar code was again reported in the 1990's. Sadly, the sources of neither the Hutcheon nor the 1990's programs were available by the late 1990's, so that problems which those programmers had at one time already solved had to be solved again. Legacy code would have been helpful.
4. To help in the design of new programs. The more issues that can be foreseen at the design stage, the less re-designing that will be needed later. For example, many structure analysts will be very pleased to be able to input space group information simply by giving the standard symbol, but occasionally users may wish either to use non-standard settings of space groups, or even invent space groups for which there is no symbol. Old programs may help broaden the horizons of new programmers.
5. Data representations. While the external representation for, say, the unit cell parameters is fairly uncontroversial, there is plenty of scope for innovation in the representation of the 'Matrix of Constraint'. A flexible program will probably offer the user a GUI for more routine tasks, and some kind of command-line alternative for more special purposes. Reviewing old programs may suggest alternative representations.

6. Finding solutions to singularities and instabilities. Careful analysis of the mathematics before the coding begins should reveal latent singularities, and it may be evident how to deal with them. Instabilities in the face of unusual data or problems are more difficult to predict, and old codes may highlight otherwise unforeseen issues.
7. Sources of efficient algorithms. While over-optimisation can hinder future development of a program, over-generalisation through lack of a proper understanding of a problem can lead to very inefficient programs. If a procedure is run frequently, a speed increase of a factor of two in what would otherwise be a 60 minute job is useful, and will still be useful even if processors themselves speed up by a factor of ten.
8. Uncovering the users *real* needs. Programs are generally written either for the authors own special needs, or in response to a need expressed by the community. In either case it is important to understand the real need, which may go beyond the immediately perceived problems.
9. Determining the scope of a new program. Writing code costs time/money. Examining similar legacy codes may help in the definition of the scope of a new program, and perhaps help with the prioritisation of the implementation of features. Leaving 'hooks' for features on a wish-list will simplify future development.
10. Assessing the cost of funding the maintenance of legacy code. Procedures such as COCOMO may help in working out the cost of producing new code to replace older codes, but they are probably not very useful for assessing the cost of maintenance and development. In these cases the design quality and documentation of the old code are probably the major influencing factors.

#### Finding Legacy Code.

Armel Le Bail has set up a Crystallography Source Code Museum in which he has simply archived FORTRAN and C programs, with their documentation if this exists in machine readable form. This is a good first-stop for software archaeology.

Yves Epelboin looks after the SinCris site, which lists alphabetically over 600 software sources. Some of these are to old codes. Unfortunately, since this is only a website of pointers to other sites, it is not uncommon for URLs to be changed without notification.

## Legacy Codes.

### Do They Have Any Value?

David Watkin  
Chemical Crystallography  
OXFORD

Siena, 2005

## Software Archaeology Legacy Codes

This talk will look at:

- The evolution of legacy codes
- Their scientific value.
- Problems with maintaining them.

### Legacy Code

Legacy Code is a generic term for software written at 'some time' in the past and falling into one of two broad categories:

#### Obsolescent.

Code currently in active use, but clearly with a limited future.

#### Obsolete.

Code no longer in active use.

### Current Code

Although there are still substantial bodies of legacy FORTRAN code still in use in the domains of both powder crystallography and macromolecular crystallography, these two fields are also active in developing modern programs.

[Small molecule crystallographers seem to worry less about using ageing software.](#)

### Obsolescent Legacy Code

In [small molecule crystallography](#), this includes **almost all** the programs currently in use.

These codes will have limited future life spans for several reasons:

1. They were written by a single author, or small group of authors.
2. They were written in languages (usually FORTRAN) which are themselves becoming obsolete.

### Obsolescent Legacy Code

Examples of obsolescent code include:

SHELX\* - based almost entirely upon the effort of one author.

SIR200\* - Designed and maintained by a non-dispersed group, and thus vulnerable to the fate of that group.

DIRDIF – The principal programmers are now retired.

CRYSTALS – The internal data structure is at the limit of adaptability.



## Obsolete Legacy Code

Obsolete legacy codes include:

- Ancestors of current obsolescent codes (e.g. SHELX-76).
- Code that the author is unable or unwilling to support.
- Codes addressing problems which have gone away (e.g. DIFABS).

## Re-inventing the Wheel

Should one try to keep old codes working, or  
should one use old codes as background to new, better, codes?

Most wheels are round  
But not all wheels are equal



[www.bedrock.deadsquid.com](http://www.bedrock.deadsquid.com)



[www.concordesst.com](http://www.concordesst.com)

## Uses of Legacy Code

Very likely, the absolute number of crystallographers able to modify and compile programs has declined since the 1970's

The number of potential users of legacy code is probably also declining.

## Uses of Legacy Code

For the few people who are able to understand and compile legacy code, potential uses are:

- Ability to perform computations not in current codes.
- Possibility of validating new codes against old codes.
- A source of information at a more detailed level than that found in published papers.

### Uses of Legacy Code - Novel Computations

Ability to perform a computations not in current codes.

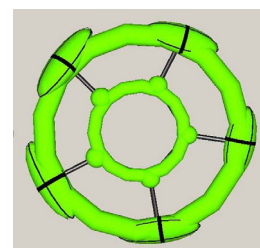
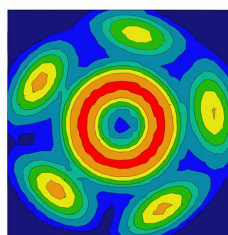
*Example:*

**Some formulas for the X-ray scattering from atoms in various spatial probability distributions.**

Murray Vernon King, and William N. Lipscomb  
*Acta Cryst.* (1950). 3, 318

### Uses of Legacy Code - Novel Computations

Example: The replacement of a disordered group of atoms by electron density distributed over a geometrical shape.



## Uses of Legacy Code - Novel Computations

### Ability to perform a computations not in current codes.

*Example:* The replacement of a disordered group of atoms by electron density distributed over a geometrical shape.

- King, M.V. and Lipscomb, W.N., (1950) Acta Cryst. 3, 155-158
- Bennett, M.J., Hutcheon, W. L. and Foxman B. M. (1975) Acta Cryst. A31, 488-494
- Chernyshev, V.V., Fetisov, G.V., Laktionov, A.V., Markov, V.T., Nesterenko, A.P. & Zhukov, S.G. (1992) J. Appl. Cryst. 25, 451 – 454
- Zlokazov, V.B. & Chernyshev, V.V. (1992) J. Appl. Cryst. 25, 447 - 451
- Chernyshev, V.V., Zhukov, S.G., Yatsenko, A.V., Aslanov, L.A. & Schenk, H. (1994) Acta Cryst. A50, 601 – 605

## Uses of Legacy Code - Novel Computations

Example: The replacement of a disordered group of atoms by electron density distributed over a geometrical shape.

By 1997 the 1975 code and the 1990's codes were unavailable for use or inspection.

None of the published descriptions of the procedure made any mention of the latent problems in refining the parameters for these shapes.

The wheel was re-invented.



## Uses of Legacy Code - Validation

It is extremely difficult to prove that a program will work correctly for any valid data input.

It is difficult to demonstrate that a program will work correctly over a wide range of unusual or marginal data inputs.

A wide user-community over a long period of time tends to uncover unstable coding.

## Uses of Legacy Code – Design

Careful examination of codes that have evolved over a long period may help in the effective design of new programs. Potential issues are:

1. Practical user requirements.
2. Data representations.
3. Singularities and instabilities.
4. Exceptions and error recovery.
5. Algorithmic efficiency.

## Uses of Legacy Code - Design

### Practical user requirements.

A programmer from a non-crystallographic background will require a very detailed specification of what the code must do.

A programmer with a crystallographic background is likely to have experience restricted to certain fields.

Old codes, or their manuals, may reveal wider requirements.

## Uses of Legacy Code - Design

### Wider user requirements.

In small-molecule crystallography it is now fashionable to input space group information in the form of the short or long symbols.

What if the user wants a non-standard setting – e.g. A -1?

What if the user wants something which cannot be represented by a conventional symbol?

## Uses of Legacy Code - Design

Wider user requirements.

*Why should the user wants a non-standard setting – e.g. A-1?*

A common reason is to simplify the visualisation of related structures – e.g. host lattices with different guests.

Space Group operators can be generated from a look-up table, or a set of rules.

## Non-Standard Settings

The layered aluminium phosphates form extended lattices able to accommodate organic guest molecules.

Pyridine complex,  $P-1$ :

$a=6.99$   $b=7.22$   $c=12.11$   $\alpha=105.1$   $\beta=104.9$   $\gamma=90.3$

Imidazole complex,  $C2/c$ :

$a=21.9$   $b=7.18$   $c=6.99$   $\alpha=90$   $\beta=104.2$   $\gamma=90$

## Non-Standard Settings

Using a  $C$  centred triclinic cell reveals the structural similarities:

Pyridine complex,  $C-1$ :

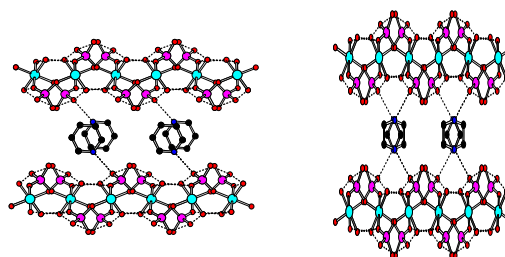
$a=23.4$   $b=7.22$   $c=6.99$   $\alpha=90.4$   $\beta=105.7$   $\gamma=88.1$

Imidazole complex,  $C2/c$ :

$a=21.9$   $b=7.18$   $c=6.99$   $\alpha=90$   $\beta=104.2$   $\gamma=90$

## Non-Standard Settings

Using a  $C$  centred triclinic cell reveals the structural similarities:



## Uses of Legacy Code - Design

Wider user requirements.

*The user wants something which cannot be represented by conventional symbols.*

In this case, if the program will not accept symmetry matrices or operators, the need cannot be fulfilled.

## Uses of Legacy Code - Design

Wider user requirements.

*Why should the user wants something which cannot be represented by conventional symbols?*

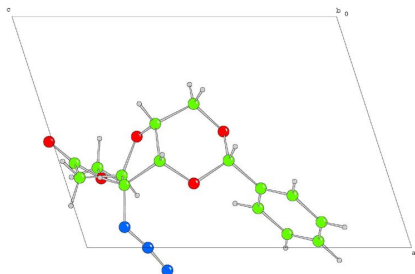
A common reason is to simplify the visualisation of related structures – e.g. structures before and after a phase transition.

## Uses of Legacy Code - Design

Wider user requirements.

Visualisation of related structures.

At 293K this sugar azide has one molecule in the asymmetric unit

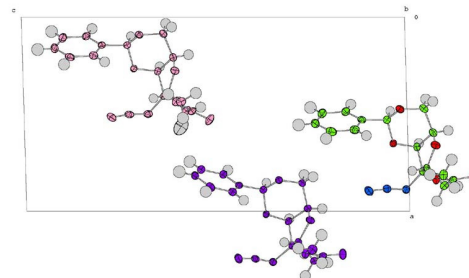


## Uses of Legacy Code - Design

Wider user requirements.

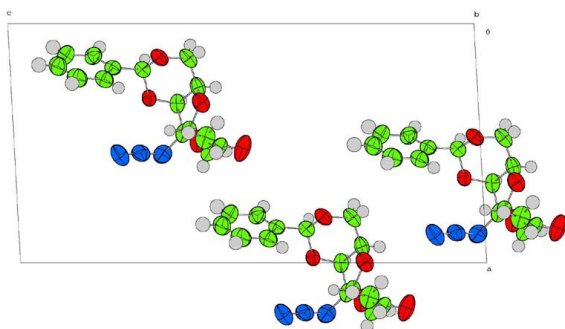
Visualisation of related structures.

At 100K the sugar azide has three molecules in the asymmetric unit



## Uses of Legacy Code - Design

Tripling the unit cell for the  $Z'=1$  cell and introducing more SG operators makes comparison of the structures simpler.



## Un-named Space Group

The true SG for both the 100K and 290K cells is  $P 2_1$ . For the tripled cell, the operators are:

SYMMETRY  $x, y, z$

SYMMETRY  $1/3+x, y, 1/3+z$

SYMMETRY  $2/3+x, y, 2/3+z$

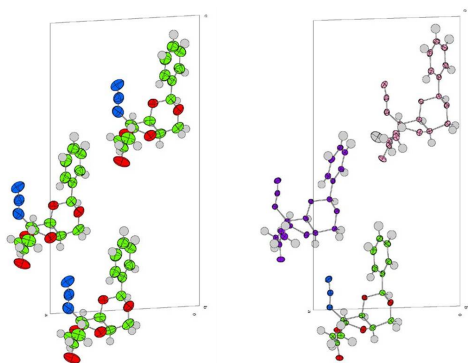
SYMMETRY  $-x, 1/2+y, -z$

SYMMETRY  $2/3-x, 1/2+y, 2/3-z$

SYMMETRY  $1/3-x, 1/2+y, 1/3-z$

## Uses of Legacy Code - Design

Tripling the unit cell and introducing more SG operators makes comparison of the structures simpler.



## Uses of Legacy Code

### Data Representations

At the design stage, attention too closely focussed on solving a particular problem may lead to restrictive data representations, and hence limited novel applications.

However, good initial design both reduces the need for future changes, and also makes the task easier if changes become inevitable.

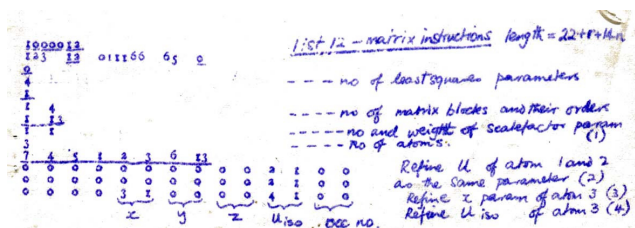
Modern languages make it easier to change data structures as a program evolves.

## Uses of Legacy Code Data Representations Evolution of CRYSTALS

'EDITION 1'	(Plain text data base definition) (Cruickshank, Freeman, Rollet, Truter, Sime, Smith and Wells, 1964)
'NOVTAPE'	(In AUTOCODE) (Hodder, Rollet, Prout and Stonebridge, Oxford, 1964),
'FAXWF'	(In ALGOL) (Ford and Rollett, Oxford, 1967),
'CRYSTALS'	(In FORTRAN) (Carruthers and Spagna, Rome, 1970)
'CRYSTALS'	(In FORTRAN, major re-write) (Carruthers, Prout, Rollet and Spagna, Oxford, 1975)
'CRYSTALS'	Issue 2 (In FORTRAN, major re-write) (Carruthers, Prout, Rollet and Watkin, Oxford 1979)
'CRYSTALS'	Issue 7 (in FORTRAN, VAX SMG user interface) (Betteridge, Prout and Watkin Oxford, 1983)
'CRYSTALS'	Issue 11 (in FORTRAN with C++ GUI) (Watkin, Prout, Carruthers, Betteridge, Cooper, 1997)

## Uses of Legacy Code Data Representations

### Evolution of CRYSTALS - [Matrix of Constraint](#) Autocode, 1965



## Uses of Legacy Code Data Representations

### Evolution of CRYSTALS - Matrix of Constraint FORTRAN, 1975

```
#LIST 12
FULL C(3,X,U[ISO])
EQUIVALENCE C(1,U[ISO]) C(2,U[ISO])
```

## Uses of Legacy Code Data Representations

### Other Solutions - SHELXL

```
FFVAR osf uiso
At1 1 10+x 10+y 10+z 11 21
At2 1 10+x 10+y 10+z 11 21
At3 2 x 10+y 10+z 11 uiso
```

## Uses of Legacy Code Data Representations

### Other Solutions – XTAL CRYLSQ

```
NOREF (X,Y,Z) (At1,At2)
NOREF (Y,Z) (At3)
CONSTR U(At1) = 1.0*U(At2)

CONSTR ps(Ats) = Q + m1p1(At1) + m2p2(At2) +
```

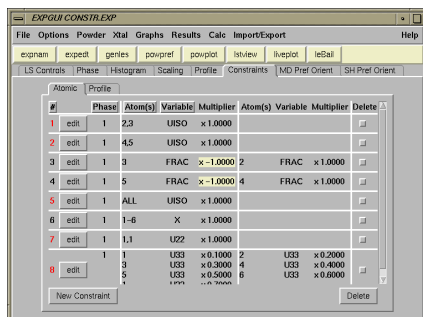
## Uses of Legacy Code Data Representations

### Other Solutions – GSAS GSAS has a coded command-line input mode

```
1 a 1
k (Konstraint)
I (Insert)
1 uiso 1 1 (phase, param, atom, mult)
1 usio 2 1
```

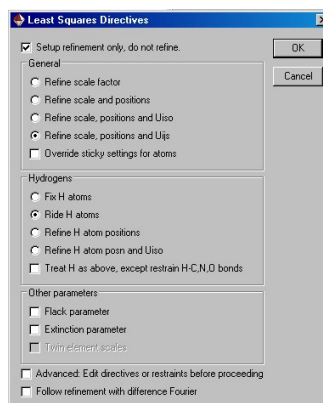
## Uses of Legacy Code Data Representations

Other Solutions - GSAS



A GUI overlay, EXPGUI, simplifies some input.

## Uses of Legacy Code



Evolution of CRYSTALS  
Matrix of Constraint

C++, 1999

## Uses of Legacy Code Data Representations

The Z' 1 & 3 structures re-visited

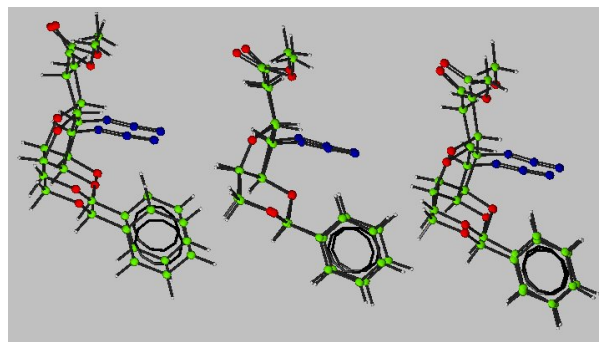
An alternative, less computationally efficient, method is to triple the contents of the asymmetric unit, and use the matrix of constraint to reduce the number of variables

```
FULL
LINK C(101,X'S) UNTIL H(1293) AND
CONT C(201,X'S) UNTIL H(2393) AND
CONT C(301,X'S) UNTIL H(3493)
```

```
LINK C(101,U'S) UNTIL N(123) AND
CONT C(201,U'S) UNTIL N(223) AND
CONT C(301,U'S) UNTIL N(323)
END
```

## Uses of Legacy Code - Design

With the 100 and 293K structures referred to a common cell and origin, the consequences of the phase change become evident



## Integral & Bolt-on GUIs

Bolt-on GUIs are generally restricted to passing normal user-commands to the program, and parsing output files.

The opportunity for real interaction is restricted.

In CRYSTALS, because we both maintain the underlying FORTRAN and designed the GUI, we can give the GUI access to anything available in the FORTRAN.

## Uses of Legacy Code - Design

### Singularities and instabilities.

Careful analysis of the mathematics before coding begins should reveal latent singularities.

e.g. standard uncertainty in a torsion angle.

$$\frac{\partial \tau}{\partial v^2} = K \left( \frac{\partial A}{\partial v^2} - \frac{A}{2B} \frac{\partial B}{\partial v^2} - \frac{A}{2C} \frac{\partial C}{\partial v^2} \right)$$

$$\frac{\partial \tau}{\partial w^2} = K \left( \frac{\partial A}{\partial w^2} - \frac{A}{2C} \frac{\partial C}{\partial w^2} \right)$$

with  $K = -1/[(BC)^{1/2} \sin \tau]$ .

Acta Cryst. (1974). A30, 848

On the standard deviation of dihedral angle. By URI SHMUELI

## Uses of Legacy Code - Design

Singularities and instabilities.

Instabilities and their cure may need  
*ad hoc* solutions.

e.g. Solution of Simultaneous Equations.

The NAG subroutine library contains 37 different routines for this purpose.

Experience with old codes may indicate which methods are most appropriate for different crystallographic tasks.

## Uses of Legacy Code - Design

Singularities and instabilities.

Instabilities and their cure may need *ad hoc* solutions.

e.g. Least Squares Parameter Refinement.

Over-shifting of ill-defined parameters can be controlled by the use of:

1. Marquardt-type augmentation of the normal matrix.
2. A matrix of partial shift (damping) factors.
3. Boundary conditions on parameter values.

## Uses of Legacy Code - Design

Algorithmic Efficiency.

Abstraction of a procedure into structured layers helps in the design, building and de-bugging of code.

However, over-generalisation may have serious impacts on performance.

Equally, over-optimisation can hinder future development of the code.

## Choosing the Right Wheel



Sometimes the programmer must make choices at the design stage, but often a better strategy is to offer a range of alternatives to the user

[www.bedrock.deadsquid.com](http://www.bedrock.deadsquid.com)

[www.concordesst.com](http://www.concordesst.com)

## User Choice

Example:

Fixing the origin in polar space groups.

e.g.  $y$  in  $P12_11$

1. Do not refine the  $y$  coordinate of one atom
2. Augment the normal matrix by using Lagrange multipliers
3. Use eigenvalue filtering of the normal matrix
4. Use a matrix of constraint
5. Use supplementary equations of restraint

## User Choice

Example: Fixing the origin in polar space groups, e.g.  $y$  in  $P12_11$

1. Do not refine the  $y$  coordinate of one atom.

This technique is available in any program which permits the user to decide which parameters to include in the refinement. The method, once popular, is now largely obsolete.

Inversion of the normal matrix by Cholesky decomposition can automatically apply the technique if the user or program fails to do anything better.



## User Choice

Example: Fixing the origin in polar space groups, e.g.  $y$  in  $P12_11$

### 2. Augment the normal matrix by using Lagrange multipliers.

This is the classical method for applying constraints to least squares, but is uncommon in widely distributed crystallographic programs.

## User Choice

Example: Fixing the origin in polar space groups, e.g.  $y$  in  $P12_11$

### 3. Use eigenvalue filtering on the normal matrix.

Eigenvalue filtering removes singularities from the normal matrix by removing degenerate parameter combinations.

It is an expensive way to fix floating origins, but the method may have other uses.

## User Choice

Example: Fixing the origin in polar space groups, e.g.  $y$  in  $P12_11$

### 4. Use a matrix of constraint.

Constraints are fundamental to refinement (for example, in dealing with atoms on special positions).

A generalised implementation gives the program user a powerful tool.

## Matrix of Constraint

The physical ('real') parameters are related to a smaller set of least squares parameters together with some additional, unconditional, knowledge.

$$[\text{physical parameters}] = [\text{knowledge}][\text{LS parameters}]$$

## Matrix of constraint

Atom on special position  $(x, -x, z)$  requires only 2 LS parameters

$$\begin{bmatrix} x \\ -x \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix}$$

## Matrix of constraint

Atom on special position  $(x, 2x, z)$  requires only 2 LS parameters.

$$\begin{bmatrix} x \\ 2x \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix}$$



### Matrix of Constraint

'Riding' a hydrogen atom on a carbon atom is done via the matrix of constraint.

$$\begin{bmatrix} C\delta x \\ C\delta y \\ C\delta z \\ H\delta x \\ H\delta y \\ H\delta z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix}$$

### Matrix of Constraint

A floating origin can be fixed via the matrix of constraint.

$$\begin{bmatrix} C\delta y_1 \\ C\delta y_2 \\ C\delta y_3 \\ C\delta y_4 \\ C\delta y_5 \\ C\delta y_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} \delta p_1 \\ \delta p_2 \\ \delta p_3 \\ \delta p_4 \\ \delta p_5 \end{bmatrix}$$

## User Choice

Example: Fixing the origin in polar space groups, e.g.  $y$  in  $P12_11$

### 5. Use supplementary Equations of Restraint

'Restrains' have become a popular method for adding knowledge into the normal matrix.

In this case the knowledge is that the scattering power weighted centre of gravity of the structure should not change during refinement.

$$0.0 = \sum_{i=1}^n w_i \delta y_i$$

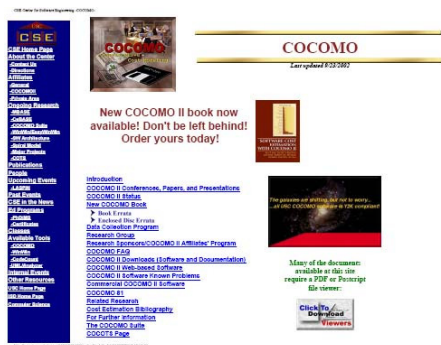
## User Choice

1. User-choice of parameters for refinement.
2. Augment the normal matrix by using Lagrange multipliers.
3. Eigenvalue filtering on the normal matrix.
4. Matrix of constraint.
5. Supplementary equations of restraint.

A problem-independent implementation of these techniques will address the floating origin issue, but also provide tools for solving other problems.

## Funding Legacy Code

## Software exists to help work out the cost of software projects



## Funding New Code

The commercial cost (including overheads, pension, insurance, salary etc) of one programmer is of the order of \$200,000 *p.a.*

If a program contains a decent amount of informative comment and is supported by both programming and user documentation, a rule of thumb costing over the whole project is:

\$20 per executable line

## The Investment in Legacy Code

The FORTRAN source of PLATON is 79,908 lines.

*The commercial cost of PLATON is about \$1.6 Million*

The FORTRAN source of the 2005 release of CRYSTALS (excluding C++ GUI and SCRIPTS) is 155,000 lines.

*The commercial cost of CRYSTALS is about \$3.1 Million*

The FORTRAN source of SHELXL is 17,134 lines

*The cost of SHELXL is \$1/3 Million??*

## Modifying Existing Programs -Effort

The cost of extending old code must include the cost of understanding it.

Much of the standard legacy software is 'economically' commented.

PLATON-01	2%
SHELXL	10%
ORFLS	11%
CRYSTALS SFLS	24%
ORION-74	33%

## Legacy Code Development Costs

The cost per line of extending old code may be much higher if:

1. The code is poorly documented.
2. The code has un-structured data management.
3. The code has an inflexible data structure.
4. Procedures are monolithic.
5. The code was optimised for speed rather than flexibility.

## Documentation

Documentation is crucial for the maintenance or recycling of old codes.

*'It is expected that the Fortran listing and the glossary of symbols which are provided will serve as a complete description of the program.'*

*ORFLS, August, 1962*

```
C      START LOOP TO STORE MATRIX AND VECTOR.
C      SEE GLOSSARY FOR STORAGE SCHEME
      DO 3010 J=1,NV
3010   DV(J)=DV(J)*SQRTW
      JK=NM
      DO 05001 J=1,NV
04301   IF(DV(J))04501,04401,04501
C       BY-PASS IF DERIVATIVE IS ZERO
04401   JK=JK-NV+J-1
      GO TO 05001
04501   DO 04801 K=J,NV
      AM(JK)=AM(JK)+DV(J)*DV(K)
      JK=JK-1
```

Sadly, the Le Bail Museum does not hold a copy of the Glossary.

## Funding Legacy Code

Funding formulae are difficult to apply to development.

Factors influencing re-implementation costs include:

- Complexity of user interface.
  - SHELX76 – *file in – file out*. 1 person-day p.o.s.
  - Main Frame CRYSTALS - *with multiple I/O files and binary data base*. 1 person-week p.o.s.
  - CRYSTALS 2005 – *with full GUI*. Unknown person-week p.o.s.

p.o.s = per operating system

## Supporting Legacy Code

Is it worth the cost and effort?

Sometimes.

People tend to like what they know.

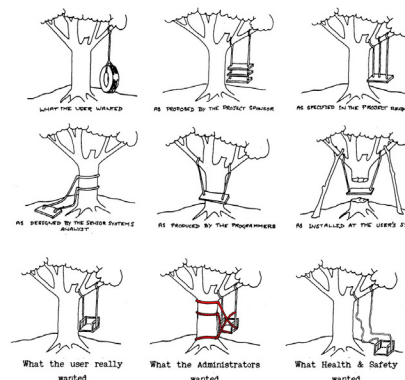
Often, people don't know what they don't know.

## Learning from the Past

Minimise the cost of developing new code.

1. Evaluate consumer's needs
2. Evaluating existing products
3. Product specification
4. Detailed design
5. Coding
6. Validation
7. Maintenance
8. Development

## Learning from the Past



Consumer's  
needs

## Learning from the Past

## Existing products – Background Research

Table 1.1

Precursors to CAMERON, 1990

NAME	Line	Ball and stick	Space filling	Polyhedra	Ellipsoids
SNOOPI	YES	YES	NO	NO	YES
PLUTO	YES	YES	NO	NO	NO
CHEM-X	YES	YES	YES	YES	NO
ORTEP	YES	NO	NO	YES	YES
SHAKAL	YES	YES	YES	NO	NO
DTMM	YES	YES	YES	NO	NO
STRUPLOT	YES	NO	NO	YES	NO
MOLDRAW	YES	YES	NO	NO	NO
COSMIC	YES	YES	NO	NO	NO
MOLVIEW	YES	YES	YES	NO	NO
MACMOLECULE	YES	YES	YES	YES	NO

# Re-use of Good Code

NORMAL80  
&  
SIR92

```

NB=8.0*ALOG10(0.05*FLOAT(MAX0(NREF,100))+0.5)
C MAXIMUM OF 30 POINTS ON WILSON PLOT
  IF(NB.GT.30) NB=30
  WRITE(NCWU,440) NREF,RHOMAX,NB
440  FORMAT(23H NUMBER OF REFLEXIONS =,I6,X,
1 32HMAXIMUM (SIN(THETA)/LAMBDA)**2 =,F7.4,X,
2 32HNUMBER OF POINTS ON WILSON PLOT =,J3)
C OBTAIN SUMS FOR WILSON PLOT AND FIT LEAST SQUARES STRAIGHT LINE
  CALL WILSUM(PTS,ISTATP,IL2BFL)
C PLOT WILSON CURVE AND LEAST SQUARES STRAIGHT LINE
  CALL GRAPH80(0,IPLOTW)
450 BT=2.0*BT
C CALCULATE SCALE FACTORS FOR APPROPRIATE REFLEXION GROUPS

      nb=8.0*alog10(0.05*float(max0(nref,100))+0.5)
c   maximum of 30 points on wilson plot
      if(nb.gt.30) nb=30
      if(ipriin.gt.0.and.jump.lt.0) write(0,442) nb
442  format(34h number of points on wilson plot =,j3)
      if(jump.ge.0) go to 450

c   obtain sums for wilson plot and fit least squares straight line
      call sir_sum(pts,ier)

      if(ier.lt.0) return
c   plot wilson and debye curves and least squares straight line
      call plotw
450 bt=2.0*bt

c   calculate scale factors for appropriate reflexion groups

```

## Finding Legacy Code

Crystallography Source Code Museum - FORTRAN

Software  
Museum.

## Le Bail.

<http://sdpd.univ-lemans.fr/museum/>

SinCris.

Y Epelboin.

This resource contains the URL of over 600 crystallography software sites.

## Finding Legacy Code

## Software database for crystallography

A	B	C	D	E	F	G	H	I
J	K	L	M	N	O	P	Q	R
S	T	U	V	W	X	Y	Z	

- ◆ [New contribution](#) to be submitted to the editor:
- ◆ [Message to the Editor](#)

Any question should be directed to the author or to the Editor  
Yves.Epelboin@lmcp.jussieu.fr.

13.01.2004 - *ShnCrEd Editor* - Copyright © International Union of Crystallography

## Finding Legacy Code

SinCris.

Some of the sites pointed to have closed down

ABSCYL	ABSEN	ABSORB	ACS
ADM	ALIGN	ALTWYCK	ADM
ANSIG	ANTHEPROT	ARITVE	ARF
ATOMS	AUTO_XPL	AXES	AZA
Acquisition of Images	Alscript	Albryk	Am
AmiraMol	Amps	Archive for MacOS	Ang
Asp	AutoDep 2.0	AutoDock	BAB
BALSAC	BEAM-Isa	BGMN	BIG
BLANC	BLAST	BOB	BRA
BRASS	BRL	BREADTH	BUD
BUSTER	bca-spreadsheets	BenchFFT	Bett
Biological software EBI	Bond Valence Wizard	Biological software JHU	Blap
CACTVS	CALCRYS	CAMEL JOCKEY	CAC
CCL	CCP14	CCP14 Solutions	CCI
CCSL	CCTBX	CELLSIZE	CEL
CGI programming	CHARMM	CHIME	CH
CF	CF2	CFLIB	CLI
CMPE	CNS	CNSsaw	CND
COMPANG	COMPDIS	CONSCRIPT	COI
CORINA	CRISP	CRUSH	CRY
CRYSCOMP-CRYSDRAW	CRYSCON	CRYSFIRE	CRY
CRYSTALVIEW	CSD	CSD2RES	CSI
CUF0UR	CVIS	Ca.R.Ine Crystallography	Ceb
l'ovus?	l'chokoll	l'chem.Rav	l'he

## Legacy Code - Conclusions

If it is well-liked and much-used, someone will maintain it.

Command-line I/O is most easily maintained.

If it has complicated I/O or a proprietary GUI, it will probably die.

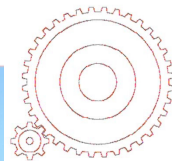
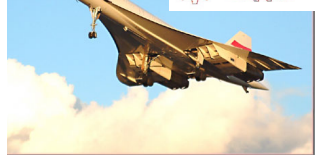
Good code can supplement published work.

Code without manuals is almost valueless.

Code with good commenting is valuable.

Don't re-invent the wheel.

Build better ones.



## Complete rewrites When, why, and how?

James W. Pflugrath<sup>1</sup>  
Rigaku/MSD, Inc., The Woodlands,  
Texas, USA

## Outline

- Previous software packages
- Current software packages
- When & Why
  - Whenever you get a benefit and the money to do it
  - Hardware requirements
  - Software requirements
- How
  - Modern software engineering practices in scientific programming
- Example
- Future software packages
- Bottom line
  - Whenever you get a benefit
  - Code maintenance

## Previous software packages

- FRODO, Alwyn Jones, 1970's
  - Evans & Sutherland PS300 version 1983

## Previous software packages

- MADNES, w/ A. Messerschmidt, 1984
  - FORTRAN77, structured (no GOTOs)
  - VAX/VMS, IRIX, Sun4, Linux, OSF1
  - Device-independent
    - Enraf-Nonius FAST, ADSC multiwire, Xentronics
  - EEC-workshops, 1980's
    - Vectorial description & algorithms, David Thomas
  - Department of Energy, Beamline X8C, E. Westbrook early 1990s

## Current software packages

- JWP moves to Molecular Structure Corporation, 1994
- d\*TREK: device-independent diffraction image processing
  - DOE subcontract, 1994, E. Westbrook
    - Simple re-write or adaptation not possible
- C++
  - Object-oriented programming language
  - No standard template library

## When & why?

- 1994
- Anytime! (2005)
- Whenever a benefit or advantage arises from the re-write
  - This is always the case, you would not make a worse piece of software would you?

## When and why?

- New programming tools
  - New languages and libraries
  - OpenGL, X Windows, OSF/Motif, Tcl/Tk
  - C++, Python
- New features
- New hardware
- New people
  - What skills do they have?
- Maintenance issues
- User issues
- Legal issues
- ~~Who pays the bills?~~

Leading With Innovation

## How?

- Write a grant
- Start a company or go to work for a company
- Start a consortium
- Make your users pay
- In other words ... sell it and get money

Leading With Innovation

## How?

- Build infrastructure
- Get computers
- Get software tools
- Get people
- Read books
- Get help

Leading With Innovation

## Software engineering practices

- Nuts & bolts
- Design beforehand
- User requirements
- Hardware requirements
- Data structures
- Algorithms
- Code management, version management
  - make, SourceSafe, cvs, bugzilla, backups
- Book: *Code Complete*
  - In the trenches: How-to
  - Variable naming, Hungarian notation

Leading With Innovation

## Example: d\*TREK

- Design submitted to DOE in late 1994
  - Data objects
    - › Devices
    - › Source, Shutter, Goniometer, Detector, Crystal
    - › Images, Reflins, Headers
    - › Interprocess communication
  - Methods
    - › Single objs: Goniometer move, Image write, etc.
    - › Multiple objs: Find, Index, Predict, Refine, Integrate, Scale/Average
    - › Reflnlist: merging, editing, sorting

Leading With Innovation

## Devices and Objects



1912



2005

Leading With Innovation



## Example: d\*TREK

- User interface
  - Simple: command line arguments
  - Scripts
  - Graphical user interface helps build command lines
    - 1994: X-Windows/Motif

## Scripting

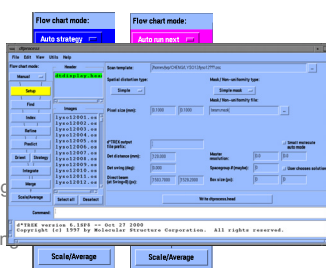
- User defaults
- High throughput
- No need for GUI
  - no button processing
- Customization
  - Beamline
  - Detector
  - Crystal

```
#!/bin/csh -f
set IMAGE_NAME = ../lyso12001.osc
set FIRST_IMAGE = 1
set LAST_IMAGE = 99

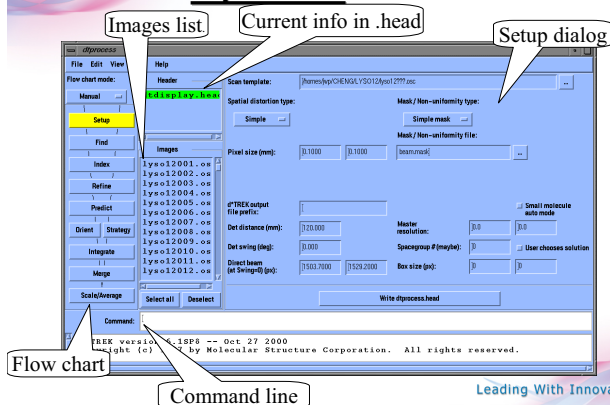
dtexttractheader $IMAGE_NAME 1.head
dtfind 1.head -seq $FIRST_IMAGE $FIRST_IMAGE -out
dtindex dtfind.head dtfind.ref
dtrefine dtindex.head dtfind.ref +All -go -go
dtrefine dtrefine.head -seq $FIRST_IMAGE +All -go
dtrefine dtrefine.head -seq $FIRST_IMAGE +All -go
dtintegrate dtrefine.head -seq $FIRST_IMAGE $LAST_
-profit -window 0 0 -batch 1 4
dtscaleaverage dtintegrate.head dtprofit.ref -sigm
-errormodel -reject .0075 \
-batchscale \
-regab spherical 4 4 \
dtscale.ref
```

## dtprocess

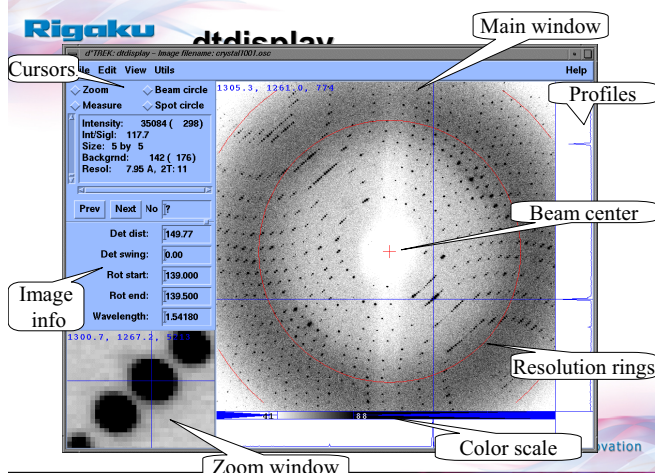
- GUI to control subprocesses
- Master scripter
- Flow chart
  - Manual
  - Auto strategy
    - one button screening
  - Auto processing
    - one button processing



## dtprocess



## dtdisplay



## Classes and objects of the d\*TREK toolkit

- Class Cimage
  - Data and methods for a 2D diffraction image
  - Several constructors
  - nRead(), nWrite(), fGetPixel(), nSetPixel(), nGetRect(), etc.
- Class Cdetector
  - Cspatial, Cnonunf, Cgoniometer
  - Cspatial::nPxToMM(), ::nMMToPX()
- Class Crefln
  - nGetHKL(), nGetH(), nGetK(), nGetL(), ...
- ...

## Classes and objects of the d\*TREK toolkit

- Class Creflnlist
  - Constructor Creflnlist() *(like Ralf's \_\_init\_\_)*
  - ::nRead()
  - ::nReduce() (needs Ccrystal object)
  - ::nSort()
  - ::nInsert(), nDelete(), nSelect()
  - ::nWrite()

## Hungarian notation

- Used at Microsoft
- Invented by a Hungarian employee of MSFT
- Examples as used in d\*TREK:

```
int    nH;
Int    *pnH;
double dH;
int    anHKL[3];
Int    a3x3dMatrix[3][3];
Ccrystal *poCrystal;
Cspacegroup *m_poSpacegroup;
```

## Hungarian notation

```
poRefln    = poGetRefln(nRefNext++);
nCentPhase = oSpacegroup.nReduceHKL(poRefln,
                                     a3nReducedHKL,
                                     &nFplusminus);
nPackedHKL = poRefln->nPackHKL(a3nReducedHKL);
```

This is controversial. But don't forget what Ralf showed us:

**template**  
can be used with float, double, int, unsigned short int, etc, so use a different

## Future software

- Current problems
  - Code maintenance in multi-platform environment
  - Lots of Windows users
    - › CrystalClear – MFC-based (native Windows GUI)
    - › Team of programmers know MFC
  - Lots of Linux users (X Windows is native to Linux)
    - › One person knows OSF/Motif
  - Lots of Mac/OSX users
  - Installation problems
  - Users know less than before
- Solution
  - Java?
  - Python?
  - wxWidgets?

## Bottom line: When, why, and how?

- There is no such thing as free software
  - At a minimum no one in this room works for free
  - "You get what you pay for." – *Harry Powell August 2005*
- Whenever there is a clear benefit
  - New hardware, operating systems
  - New users
  - New programmers
  - New methods
- ...

## Bottom line: When, why, and how?

- Whenever you can get money to do it
  - Consortia
  - Beamlines
  - Charge for-profit companies
- How:
  - Get serious about software engineering practices
  - Read books
  - Take classes
  - Hire staff – give them a stake in it



### One last thing ...

“Remember, software is just like paper:  
It’s the result of research.”

--- *Wladek Minor, May 29, 2005*

“Software is just like toilet paper: Users  
want to use the softest available, then  
throw it away.”

--- *Jim Pflugrath, June 1, 2005*

### Acknowledgements

- Rigaku/MSC
  - Thad Niemeyer
  - Robert Bolotovskiy
  - Cheng Yang
  - Kris Tesh
  - Tom Hendrixson
  - Joe Ferrara
- Ed Westbrook
- R. Jacobson
- US Dept of Energy
  - Contract 943072401
- Gerard Bricogne
  - EEC Workshops
- Clemens Vonnrhein

# Coordinate Systems

Coordinate systems, operators, and transformations.

Kevin Cowtan  
cowtan@ysbl.york.ac.uk

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Coordinate Systems

## Coordinate systems

- We have to deal with many coordinate systems in crystallographic software. The book-keeping is not exciting, but it is vital. Simplifications (e.g. assuming orthogonal grids) must usually be paid for later.
- Examples:
  - Orthogonal Ångstrom coordinates.
  - Fractional coordinates.
  - Reciprocal orthogonal coordinates.
  - Reflection (Miller) indices, i.e. *HKLs*.
  - Grid coordinates.

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Coordinate Systems

## Coordinate Systems

- In addition, we need to handle:
  - Transformations between these coordinate systems.
  - Transformations within a coordinate system (i.e. rotation and translation operators).
  - Rotation representations.
  - Derivatives of functions with respect to different coordinate systems.
- This lecture will give a basic overview of the issues. Implementations of all these data types and transformations are a part of both the **CCTBX** and **Clipper** crystallographic libraries.

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Coordinate Systems

## Coordinate Systems

- Conventions for this lecture:
  - scalars are italic, lower case, e.g. *s*, *r*
  - vectors are italic, lower case and underlined, e.g.  $\underline{x}$ ,  $\underline{y}$ ,  $\underline{h}$
  - matrices are italic, uppercase and bold, e.g. ***O***, ***F***, ***M***
- In addition to matrix notation, most equations are also given as explicit sums of terms.
  - The elements of a vector or matrix are given by the same symbol in italic lower case with an appropriate number of subscripts. e.g.  $x_i$ ,  $O_{ij}$ .
- All vectors and matrices are of rank 3.

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Coordinate Systems

## Coordinate Systems

Coordinate systems:

- Coordinate are used to describe the positions of elements within the crystal system. e.g. the position of a particular atom within a unit cell, or a particular grid point within a grid.
- 3 dimensions -> rank 3.
  - Coordinates represented by a vector of 3 numbers.

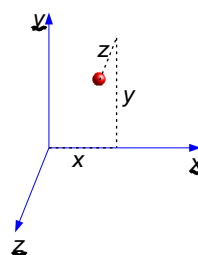
Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Coordinate Systems

## Coordinate Systems: Real space

Orthogonal Ångstrom Coordinates:

- 3 orthogonal distances in Ångstroms along directions  $\underline{x}$ ,  $\underline{y}$ ,  $\underline{z}$  (Formally: basis vectors)



$$\underline{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Note: there is no reference to the unit cell at this point.

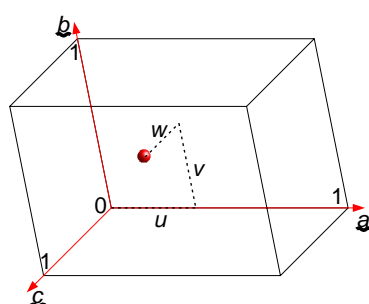
Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Coordinate Systems

## Coordinate Systems: Real space

Fractional coordinates:

- Position in the unit cell described as a fractional position along each cell edge:



$$\underline{u} = \begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$$

Note: since the cell repeats,  $u, v, w$  repeat on the range 0...1. We often standardize on the range 0...1 (or -1/2...1/2), but this may split a molecule.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Real space

Relating orthogonal and fractional coordinates:

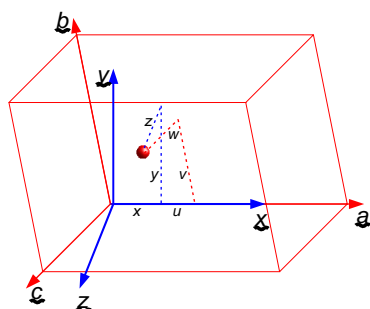
- We can orient and position one coordinate system however we want with respect to the other, *but...*
- It is convenient to adopt some convenient convention.
- The most common convention in the PDB (also the CCP4 default) is:
  - Align the  $\underline{a}$  axis along  $\underline{x}$
  - Align the  $\underline{b}$  axis in the  $\underline{x}$ - $\underline{y}$  plane
    - Or equivalently align  $\underline{c}$  axis perpendicular to  $\underline{a}$  and  $\underline{b}$

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Real space

Relating orthogonal and fractional coordinates:



Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Real space

Relating orthogonal and fractional coordinates:

- An orthogonal coordinate may be determined from a fractional coordinate by:

$$\underline{x} = \underline{O} \underline{u}$$

i.e.

$$\underline{x} = \underline{O} \underline{u}$$

$$x_i = \sum_j O_{ij} u_j$$

Where  $\underline{O}$  is the orthogonalization matrix. For the common convention,

$$\underline{O} = \begin{pmatrix} a & b \cos(\gamma) & c \cos(\beta) \\ 0 & b \sin(\gamma) & -c \sin(\beta) \cos(\alpha^*) \\ 0 & 0 & c \sin(\beta) \sin(\alpha^*) \end{pmatrix}$$

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Real space

Relating orthogonal and fractional coordinates:

- A fractional coordinate may be determined from an orthogonal coordinate by:

$$\underline{u} = \underline{F} \underline{x}$$

i.e.

$$\underline{u} = \underline{F} \underline{x}$$

$$u_i = \sum_j F_{ij} x_j$$

Where  $\underline{F}$  is the fractionalization matrix.

Clearly  $\underline{F} = \underline{O}^{-1}$

- Note:
  - $a, b, c, \alpha, \beta, \gamma$  are the cell constants.
  - $a^*, b^*, c^*, \alpha^*, \beta^*, \gamma^*$  are the reciprocal cell constants.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Real space

Measuring distances:

- We do this all the time, e.g. inter-atomic distances.
- In orthogonal coordinates, the squared distance between two points is given by  $r^2 = \Delta x^2 + \Delta y^2 + \Delta z^2$

$$\underline{\Delta x} = \underline{x}_2 - \underline{x}_1$$

$$r^2 = \underline{\Delta x}^T \underline{\Delta x}$$

or:

$$r^2 = \sum_i \Delta x_i^2$$

$$r^2 = \underline{x}^T \underline{x}$$

## Coordinate Systems: Real space

Measuring distances:

- For the distance between two fractional coordinates, convert to orthogonal first:

$$r^2 = \underline{\Delta u}^T \mathbf{O}^T \mathbf{O} \underline{\Delta u}$$

or:

$$r^2 = \sum_i \sum_j \sum_k O_{ij} O_{ik} \Delta u_j \Delta u_k$$

- Simplify by pre-calculating the central product:

$$\mathbf{M} = \mathbf{O}^T \mathbf{O}$$

$$M_{jk} = \sum_i O_{ij} O_{ik}$$

- $\mathbf{M}$  is a symmetric matrix, called the "real-space metric tensor".

$$r^2 = \underline{u}^T \mathbf{O}^T \mathbf{O} \underline{u}$$

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Real space

Measuring distances:

- Simplified form using the metric tensor:

$$r^2 = M_{11} \Delta u_1^2 + M_{22} \Delta u_2^2 + M_{33} \Delta u_3^2 + 2 M_{12} \Delta u_1 \Delta u_2 + 2 M_{13} \Delta u_1 \Delta u_3 + 2 M_{23} \Delta u_2 \Delta u_3$$

(since the matrix is symmetric, we just use the upper triangle and double the off-diagonal terms).

- (This is often a performance critical task).

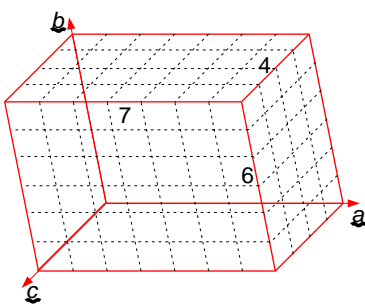
Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Real space

Other coordinate types:

- Grid coordinates: Electron density maps are usually calculated on a grid which samples the unit cell



$$\underline{n} = \begin{pmatrix} n_u \\ n_v \\ n_w \end{pmatrix} = \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix}$$

$$\text{e.g. } \underline{n} = \begin{pmatrix} 7 \\ 6 \\ 4 \end{pmatrix}$$

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Real space

Other coordinate types:

- Grid coordinates:  $\underline{g}$

- Each grid point is indexed using 3 integer indices,  $\underline{g}$ , usually starting from 0.
- The sampling usually involves a grid in which each cell edge is divided into a set number of equal divisions, with the number of divisions roughly proportional to the cell edge.
- Symmetry and FFT requirements may constrain these values (e.g. multiple of 2, 3, 4, no large prime factors).

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Real space

Other coordinate types:

- Grid coordinates:

- Convert to grid coordinates by scaling the fractional coordinates by the samplings, and taking nearest integer:

$$g_i = \text{int}(n_i u_i)$$

$$u_i = g_i / n_i$$

- For orthogonal coordinates, convert to/from fractional first. As an optimization, the two steps can be combined.
- Grid coordinates repeat every  $n_i$  along the  $i$ th axis.

## Coordinate Systems: Real space

Other coordinate types:

- Grid coordinates: Additional complications:

- M. Rowicka, A. Kudlicki and Z. Otwinowski, [Acta Cryst. (2002). A58, 574-579] use grids which do not intersect the origin to improve symmetry handling in the FFT
- Hexagonal close packed grids give a more efficient sampling of real space. How are they best indexed?

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Real space

Other coordinate types:

- Map coordinates: (Cowtan)
  - Non-integer grid coordinates.
    - For crystallographic maps, fractional coordinates do the job just fine
    - For non-crystallographic maps, fractional coordinates are undefined.
  - Used for interpolation in non-crystallographic maps.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Real space

Implementation issues:

- It is very easy when programming to make mistakes over coordinate types.
- When using strongly typed languages (e.g. C++), implement each coordinate type as a different class to prevent such errors.
  - (Use inheritance for common behaviors)
- Coordinate types or cell and sampling classes may then implement all the required conversions.

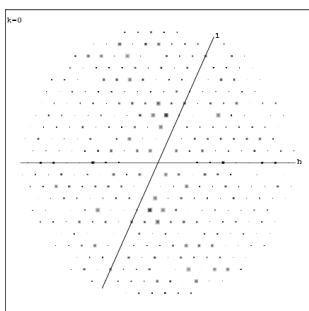
See **CCTBX** or **Clipper**.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Reciprocal space

In reciprocal space we mainly deal with reflections, indexed by integer  $h, k, l$ .



$$\underline{h} = \begin{pmatrix} h \\ k \\ l \end{pmatrix} = \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix}$$

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Reciprocal space

In reciprocal space we mainly deal with reflections, indexed by integer  $h, k, l$ .

- $h, k, l$  are coordinates on a non-orthogonal grid, like grid coordinates.
- $h, k, l$  do not repeat. They are centered (+/-) about the origin  $h=k=l=0$ .
- May be referred to as Miller indices (by correspondence with the indexing of crystal cell faces).

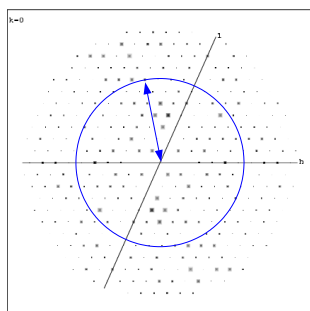
Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Reciprocal space

We frequently need to determine the (reciprocal) distance from a reflection to the origin (resolution).

We may also need to determine a reciprocal orthogonal coordinate e.g. to make this picture.



Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Reciprocal space

Relating reciprocal orthogonal and fractional coordinates:

- An reciprocal orthogonal coordinate may be determined from an HKL by:

$$\underline{s} = \underline{O}^* \underline{h}$$

- As in real space,  $\underline{O}^*$  is determined by the orthogonalization convention. A convenient choice is to use the transpose of the real space fractionalizing matrix:  $\underline{O}^* = \underline{F}^T$ 
  - i.e.  $\underline{z}$  parallel to  $\underline{c}^*$ ,  $\underline{y}$  in the  $\underline{b}^*-\underline{c}^*$  plane

## Coordinate Systems: Reciprocal space

Relating reciprocal orthogonal and fractional coordinates:

- For display purposes, the “Cambridge Convention” is more common:
  - $\underline{x}$  parallel to  $\underline{a}^*$
  - $\underline{y}$  in the  $\underline{a}^*-\underline{b}^*$  plane
- In this case  $\mathbf{O}^*$  is calculated using the equivalent formula to  $\mathbf{O}$ .
- Don't mix conventions between real and reciprocal space – it will only end in tears.**

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Reciprocal space

Measuring distances in reciprocal space:

- As before, to calculate squared distances (in inverse squared Ångströms), we first need reciprocal orthogonal coordinates, or a reciprocal metric tensor:

$$s^2 = \underline{h}^T \mathbf{O}^{*T} \mathbf{O}^* \underline{h} \quad \boxed{s^2 = \underline{h}^T \mathbf{O}^{*T} \mathbf{O}^* \underline{h}}$$

or:

$$s^2 = \sum_i \sum_j \sum_k \mathbf{O}^*_{ij} \mathbf{O}^*_{ik} h_j h_k$$

- Simplify by pre-calculating the central product:

$$\mathbf{M}^* = \mathbf{O}^{*T} \mathbf{O}^*$$

$$M^*_{jk} = \sum_i \mathbf{O}^*_{ij} \mathbf{O}^*_{ik}$$

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Reciprocal space

Measuring distances:

- Simplified form using the metric tensor:
 
$$s^2 = M^*_{11} h^2 + M^*_{22} k^2 + M^*_{33} l^2 + 2 M^*_{12} h k + 2 M^*_{13} h l + 2 M^*_{23} k l$$
- Resolution in Ångströms is  $1/s$ , i.e.  $\sqrt{1/s^2}$
- $s^2 = 4 \sin^2 \theta / \lambda^2$
- Some developers use the symbol  $s$  instead of  $s^2$ .
- Clipper and CCP4 refer to  $s^2$  as inverse resolution squared: “invresolsq”.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

### Interlude

The Structure Factor Equation:

- From Scattering Theory:
 
$$F(\underline{s}) = \sum_j f_j(\underline{s}) \exp(2\pi i \underline{s}^T \underline{x})$$
- Structure Factor Equation:
 
$$F(\underline{h}) = \sum_j f_j(\underline{s}) \exp(2\pi i \underline{h}^T \underline{u})$$
- Because:
 
$$\begin{aligned} \underline{s}^T \underline{x} &= (\mathbf{F}^T \underline{h})^T (\mathbf{O} \underline{u}) \\ &= \underline{h}^T \mathbf{F} \mathbf{O} \underline{u} \\ &= \underline{h}^T \underline{u} \end{aligned}$$

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Operators

Operators transform coordinates in such a way that a rigid body will moved to a new position within the coordinate system. We consider three types:

- Translation operators:
  - Move an object without rotating it.
- Rotation operators:
  - Rotate an object about the origin of the coordinate system.
- Rotation-translation (RT) operators:
  - Rotate and translate an object, or equivalently, rotate an object about a point other than the origin.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Operators

Translation operators:

- Add the translation vector to the existing coordinate to get the new coordinate (in the same system).

$$\underline{x}_2 = \underline{x}_1 + \underline{T}_x$$

$$\underline{u}_2 = \underline{u}_1 + \underline{T}_u$$

- Translation vectors transform like coordinates, i.e. if  $\underline{x}_1 = \mathbf{O} \underline{u}_1$  and  $\underline{x}_2 = \mathbf{O} \underline{u}_2$  then:

$$\underline{T}_x = \mathbf{O} \underline{T}_u$$

$$\underline{T}_u = \mathbf{F} \underline{T}_x$$

- The inverse of  $\underline{T}$  is  $-\underline{T}$ .

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

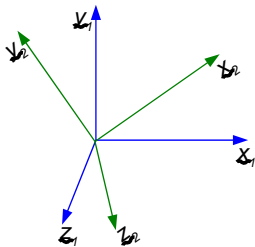
Sienna/Coordinate Systems

## Coordinate Systems: Operators

Rotation operators:

- A rotation is described by a matrix  $\mathbf{R}$  which is orthonormal (i.e.  $\mathbf{R}^{-1} = \mathbf{R}^T$ )

$$\underline{x}_2 = \mathbf{R}_x \underline{x}_1$$



$$\mathbf{R}_x = \begin{pmatrix} \cos(\phi_{X_1 X_2}) & \cos(\phi_{X_1 Y_2}) & \cos(\phi_{X_1 Z_2}) \\ \cos(\phi_{Y_1 X_2}) & \cos(\phi_{Y_1 Y_2}) & \cos(\phi_{Y_1 Z_2}) \\ \cos(\phi_{Z_1 X_2}) & \cos(\phi_{Z_1 Y_2}) & \cos(\phi_{Z_1 Z_2}) \end{pmatrix}$$

Kevin Cowtan, [cowntan@ysbl.york.ac.uk](mailto:cowntan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Operators

Rotation operators:

- We can represent the rotation in fractional coords:

$$\underline{x}_2 = \mathbf{R}_x \underline{x}_1$$

$$\underline{u}_2 = \mathbf{R}_u \underline{u}_1$$

$$\text{Since } \underline{x}_1 = \mathbf{O} \underline{u}_1 \text{ and } \underline{x}_2 = \mathbf{O} \underline{u}_2, \quad \mathbf{O} \underline{u}_2 = \mathbf{R}_x \mathbf{O} \underline{u}_1$$

therefore:

$$\mathbf{R}_u = \mathbf{O}^{-1} \mathbf{R}_x \mathbf{O}$$

and:

$$\mathbf{R}_x = \mathbf{O} \mathbf{R}_u \mathbf{O}^{-1}$$

$$\mathbf{R}_u = \mathbf{O}^{-1} \mathbf{R}_x \mathbf{O}$$

$$\mathbf{R}_x = \mathbf{O} \mathbf{R}_u \mathbf{O}^{-1}$$

- Note:  $\mathbf{R}_u$  is **not** a rotation matrix. ( $\mathbf{R}_u^{-1} \neq \mathbf{R}_u^T$ )

Kevin Cowtan, [cowntan@ysbl.york.ac.uk](mailto:cowntan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Operators

Rotation-translation (RT) operators:

- A rotation-translation operator is described by a rotation matrix  $\mathbf{R}$  followed by a translation  $\underline{I}$ .

$$\underline{x}_2 = \mathbf{R}_x \underline{x}_1 + \underline{I}_x$$

- We can represent this as a single vector **operator**:

$$\underline{x}_2 = \mathbf{R}_x(\underline{x}_1)$$

- Its inverse is given by the rotation  $\mathbf{R}_x^{-1}$  and the translation  $-\mathbf{R}_x^{-1} \underline{I}_x$ :

$$\underline{x}_1 = \mathbf{R}_x^{-1} (\underline{x}_2 - \underline{I}_x) = \mathbf{R}_x^{-1} \underline{x}_2 - \mathbf{R}_x^{-1} \underline{I}_x$$

- To convert to fractional, convert  $\mathbf{R}$  and  $\underline{I}$  as before.

Kevin Cowtan, [cowntan@ysbl.york.ac.uk](mailto:cowntan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Operators

Rotation-translation (RT) operators:

- Implementation:

- Use a class to hold the rotation matrix and translation vector, and provide methods to transform a vector, invert, and convert the operator to other coordinate systems.
- Historical: Fortran 77 does not support classes, so developers often grouped the rotation and translation in a 3x4 or 4x4 matrix. Mathematically, vectors must be augmented to length 4 by appending a '1'.

Kevin Cowtan, [cowntan@ysbl.york.ac.uk](mailto:cowntan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Rotations

Rotations have many representations:

- Matrix:
  - use directly to manipulate vectors
  - uniquely defined, 9 numbers ( $\underline{m}_{ij}$ ).
- Quaternions:
  - uniquely defined, 4 numbers ( $x, y, z, w$ ).
- Euler angles:
  - multiple conventions, 3 numbers ( $\alpha, \beta, \gamma$ ).
- Polar angles:
  - multiple conventions, 3 numbers ( $\phi, \psi, \kappa$ ).

Kevin Cowtan, [cowntan@ysbl.york.ac.uk](mailto:cowntan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Rotations

Quaternions:

- 4 numbers: ( $x, y, z, w$ )
  - $x, y, z$  are the direction cosines of the rotation axis, scaled by  $\sin(\kappa/2)$
  - $w$  gives the angle of rotation, as  $\cos(\kappa/2)$ .
- No ambiguity in definition.
- Easy to convert to Matrix, Euler, Polar
  - good as an interchange format. See **Clipper**, rotation.cpp

Kevin Cowtan, [cowntan@ysbl.york.ac.uk](mailto:cowntan@ysbl.york.ac.uk)

Sienna/Coordinate Systems

## Coordinate Systems: Rotations

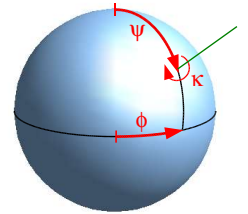
Euler angles:

- 3 numbers:  $(\alpha, \beta, \gamma)$ .
  - $\alpha$  is rotation about  $z$
  - $\beta$  is rotation about new  $y$
  - $\gamma$  is rotation about new  $z$
- 24 conventions (which axis to rotate about, stationary or moving axes), but crystallographers all uses **ZYZr**, so well standardised.
- Convenient for rotation function search limits.
- Convenient for program input.

## Coordinate Systems: Rotations

Polar angles:

- 3 numbers:  $(\phi, \psi, \kappa)$  or  $(\omega, \phi, \kappa)$
- $(\phi, \psi)$  define the direction of the axis,  $\kappa$  is the angle of rotation about it.
- Easy to understand.
- Inconsistent conventions.
- Use for program output only.



## Coordinate Systems: Derivatives

Many calculations require that we calculate derivatives of some function with respect to some coordinate. e.g.

- Refinement:
  - Refinement of individual atomic coordinates and B-factors:  $(x_j, B_j)$
- Molecular replacement:
  - Rigid body refinement of search model against density:  $(R_x, T_x)$ .

## Coordinate Systems: Derivatives

- Refinement, MR both depend on calculation of density gradients and curvatures – these are calculated along grid/cell directions, i.e. fractional coordinates.
- Rigid body rotations, and refinement restraints (e.g. bond lengths/angles), are calculated using orthogonal coordinates.
- Need to convert derivatives between fractional and orthogonal.

## Coordinate Systems: Derivatives

- e.g. for density gradients,  
 $f = \rho$

$$g_u = \begin{pmatrix} \frac{\partial f}{\partial u} \\ \frac{\partial f}{\partial v} \\ \frac{\partial f}{\partial w} \end{pmatrix} \quad c_u = \begin{pmatrix} \frac{\partial^2 f}{\partial u^2} & \frac{\partial^2 f}{\partial u \partial v} & \frac{\partial^2 f}{\partial u \partial w} \\ \frac{\partial^2 f}{\partial v \partial u} & \frac{\partial^2 f}{\partial v^2} & \frac{\partial^2 f}{\partial v \partial w} \\ \frac{\partial^2 f}{\partial w \partial u} & \frac{\partial^2 f}{\partial w \partial v} & \frac{\partial^2 f}{\partial w^2} \end{pmatrix}$$

$$g_x = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{pmatrix} \quad c_x = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z^2} \end{pmatrix}$$

## Coordinate Systems: Derivatives

- Gradients transform using the transpose of the inverse matrix (because the coordinate is in the denominator):

$$g_u = O^T g_x$$

$$g_x = F^T g_u$$

$$g_{u,j} = \sum_i O_{ij} g_{x,i} \quad g_{x,j} = \sum_i F_{ij} g_{u,i}$$

- Curvatures:

$$c_u = O^T c_x O$$

$$c_x = F^T c_u F$$

$$C_{u,kl} = \sum_i \sum_j O_{ik} O_{jl} C_{x,ij} \quad C_{x,kl} = \sum_i \sum_j F_{ik} F_{jl} C_{u,ij}$$



# Coordinate Systems

## Summary:

- In crystallographic calculations, we need to use a range of coordinate systems:
  - real and reciprocal space
  - orthogonal, fractional, and grid.
- We also use operators in each space.
  - rotations, translations, and RT.
- Coordinates and operators are related by orthogonalising and fractionalising matrices and their transposes in various combinations.

# The PHENIX project



Crystallographic software for automated structure determination

**Computational Crystallography Initiative (LBNL)**  
-Paul Adams, Ralf Grosse-Kunstleve, Peter Zwart, Nigel Moriarty, Nicholas Sauter, Pavel Afonine



**Los Alamos National Lab (LANL)**  
-Tom Terwilliger, Li-Wei Hung, Thiru Radhakannan



**Cambridge University**  
-Randy Read, Airlie McCoy, Laurent Storoni, Hamsapriye



**Texas A&M University**  
-Tom Ioerger, Jim Sacchettini, Kreshna Gopal, Lalji Kanbi, Erik McKee, Tod Romo, Reetal Pai, Kevin Childs, Vinod R



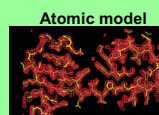
## Structure Determination by MAD/SAD/MIR in PHENIX

**PROVIDED THAT:**

Data files (H K L Fobs Sigma)  
Crystal information (Space group, Cell)  
Scattering factors (for MAD)

Data are strong, accurate, < 3 Å  
Strong anomalous signal  
Little decay  
Space group is correct  
Scattering factors close (for MAD data)  
You are willing to wait a little while...  
(10 minutes to hours, depending on size)

Heavy-atom coordinates  
Non-crystallographic symmetry  
Electron-density map



Model is 50-95% complete  
(depending on resolution)

Model is (mostly) compatible with the data...but is not completely correct

Model requires manual rebuilding  
Model requires validation and error analysis

## Major needs in automated structure solution

### MAD/SAD/MIR

Robust structure determination procedures  
Best possible electron density maps to build most complete model  
Decision-making about best path for structure solution

### Molecular Replacement

Use of distant models  
Preventing model bias

### All structures

Model completion/Ligand fitting  
Error analysis  
Decision-making for what data to use and what path to follow  
How to incorporate vast experience of crystallographic community

## Best possible electron density maps to build the most complete model

**Statistical density modification**

**Local patterns of density**

**ID of fragments**

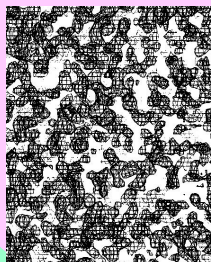
**Iterative model-building and refinement**

**FULL-OMIT density modification and model-building**

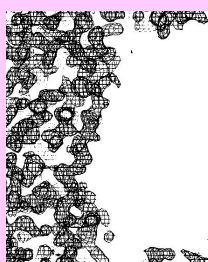
## Why we need good measures of the quality of an electron-density map:

Which solution is best?

Are we on the right track?



If map is good:  
It is easy



## Statistical density modification

(A framework that separates map information from experimental information and builds on density modification procedures developed by Wang, Brice, and others)

• Principle: phase probability information from probability of the map and from experiment:

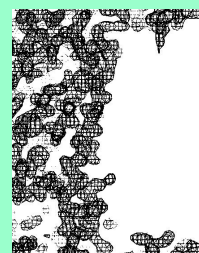
$$P(\phi) = P_{\text{map probability}}(\phi) P_{\text{experiment}}(\phi)$$

• "Phases that lead to a believable map are more probable than those that do not"

• A believable map is a map that has...

- a relatively flat solvent region
- NCS (if appropriate)
- A distribution of densities like those of model proteins

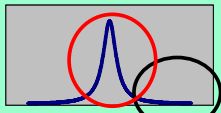
• Calculating map probability ( $\phi$ ):  
• calculate how map probability varies with electron density  $p$   
• Use chain rule to deduce how map probability varies with phase (equations of Brice, 1992).



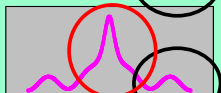
### Map probability phasing:

Getting a new probability distribution for a single phase given estimates of all others

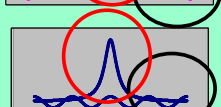
1. Identify expected features of map (flat far from center)
2. Calculate map with current estimates of all structure factors except one (k)
3. Test all possible phases  $\phi$  for structure factor k (for each phase, calculate new map including k)
4. Probability of phase  $\phi$  estimated from agreement of maps with expectations



A function that is (relatively) flat far from the origin



Function calculated from estimates of all structure factors but one (k)



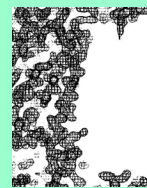
Test each possible phase of structure factor k.  $P(\phi)$  is high for phase that leads to flat region



### A map-probability function

Log-probability of the map is sum over all points in map of local log-probability

$$LL^{MAP}(\{\mathbf{F}_h\}) \approx \frac{N_{REF}}{V} \int_V LL(\rho(\mathbf{x}), \{\mathbf{F}_h\}) d^3\mathbf{x}$$



A map with a flat (blank) solvent region is a likely map

Local log-probability is believability of the value of electron density ( $p(\mathbf{x})$ ) found at this point

$$LL(\rho(\mathbf{x}), \{\mathbf{F}_h\}) = \ln[p(\rho(\mathbf{x})|PROT)p_{PROT}(\mathbf{x}) + p(\rho(\mathbf{x})|SOLV)p_{SOLV}(\mathbf{x})]$$

If the point is in the PROTEIN region, most values of electron density ( $p(\mathbf{x})$ ) are believable

If the point is in the SOLVENT region, only values of electron density near zero are believable

### Statistical density modification features and applications

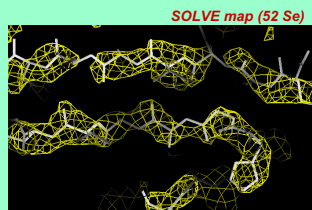
#### Features:

- Can make use of any expectations about the map.
- A separate probability distribution for electron density can be calculated for every point in the map

#### Applications:

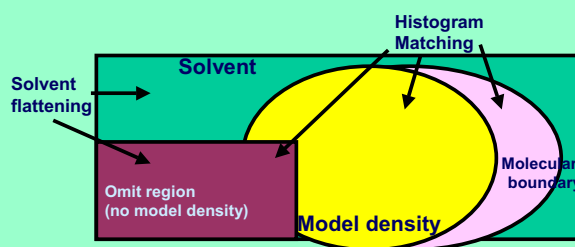
- Solvent flattening
- Non-crystallographic symmetry averaging
- Template matching
- Partial model phasing
- Prime-and-switch phasing
- General phase recovery
- Iterative model-building

Reference: Terwilliger, T. C. (2000), Maximum-likelihood density modification. Acta Crystallographica, D55, 1863-1871.



### Composite omit map with statistical density modification

Statistical density modification allows a separate probability distribution for electron density at each point in the map: can specify that "missing" density is within molecular boundary



Can be used with or without experimental phases...with or without omit

### Image enhancement using local feature recognition

Electron density maps of proteins have many features in common

- Connected density
- Preferred distances for spacing between regions of high density
- Preferred shapes of density

Starting image in red

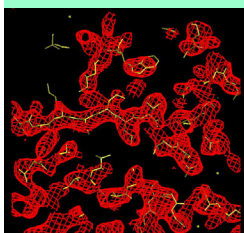
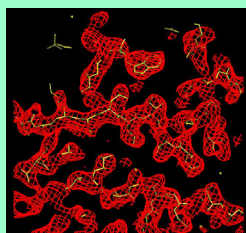


Image improved using expectations about local features



### Image enhancement using local feature recognition

Approach:

- Use the pattern of density near a point x to estimate the value of density at x
  - Combine new estimate of density with previous one to improve the overall image
- "Local NCS averaging"

Starting image in red

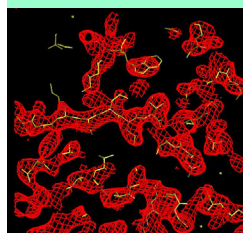
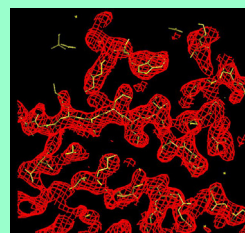


Image improved using expectations about local features



### Image enhancement using local feature recognition

#### Approach:

#### •Create $N$ templates of local density using model data

- Examine density near each point  $x$  in image (within 2 Å)
- Exclude region very close to  $x$  (about 1 Å)
- Cluster and average local patterns of density (after rotation to maximize CC)

#### •Identify relationship between finding pattern $k$ of density near $x$ , and density at $x$

- Find all locations in the image where template  $k$  best matches the local density near  $x$
- Calculate average value of density at  $x$  for these cases =  $\rho_{\text{mean}}(k)$

#### •Identify pattern near each point in actual map and use it to estimate density at that point

- For each point  $x$  in the image, identify which template  $k$  best matches the local density near  $x$
- Use  $\rho_{\text{mean}}(k)$  as estimate of density at  $x$

### Image enhancement using local feature recognition

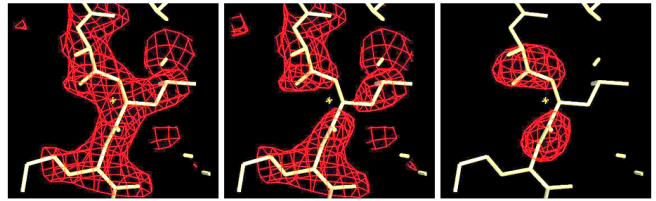
Remove all information about density at  $x$  from  $\rho(x + \Delta x)$

$\rightarrow g(x + \Delta x)$ , unbiased estimate of local pattern at  $x$

Select most similar template  $k$  from library of unbiased patterns

Generate new estimate of density at  $x$  from average value at center of template  $k$

A template associated with positive density...



$\rho_{\text{current}}(x + \Delta x)$

Local density in current map

$g(x + \Delta x)$

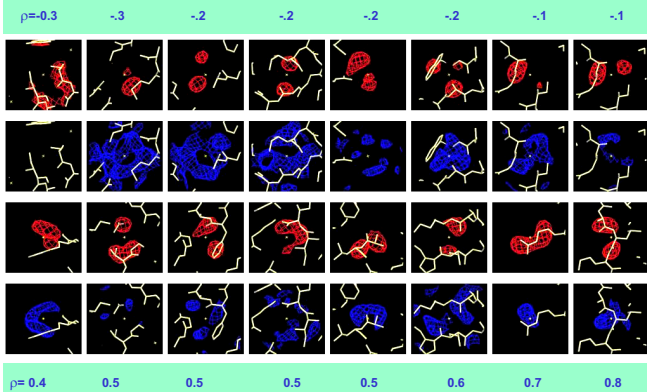
Bias-removed local density...subtract  $\rho_{\text{current}}(x)$  convoluted with origin of Patterson from all nearby points

$t(\Delta x)$

Closest template in library (after testing 168 rotations)  
 $\langle \rho \rangle$  for this template : 0.8 +/- 0.9

### Image enhancement using local feature recognition

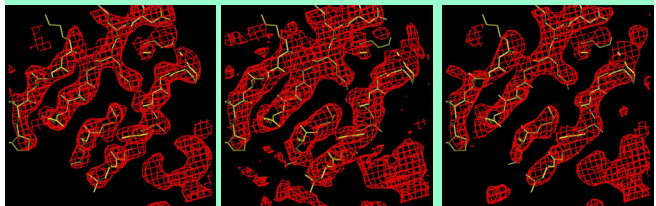
Templates associated with low density (top rows) and high density (bottom rows)  
RED=positive contours BLUE=negative contours for the same template



### Image enhancement using local feature recognition

Image recovery from a good map...

- gives an image that has (mostly) correct features
- errors are (almost) uncorrelated with original errors



RESOLVE map gene 5 protein at 2.6 Å

CC to perfect map = 0.8

Recovered image derived from RESOLVE map

CC to perfect map = 0.36

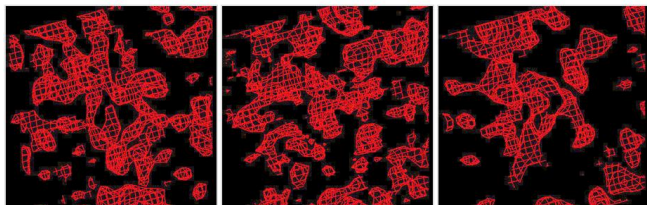
Map phased using only information from recovered image

CC to perfect map = 0.64

CC of errors with errors in RESOLVE map = 0.11

### Image enhancement using local feature recognition

Image recovery from a random map...gives an uncorrelated image



Random map at 2.6 Å

Recovered image derived from random map

CC to original random map=-0.01

Map phased using only recovered image

CC to original random map=-0.04

### Iterative procedure for image enhancement using local feature recognition

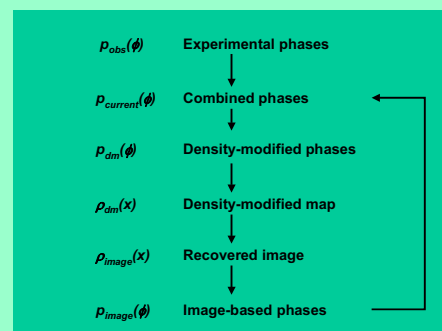
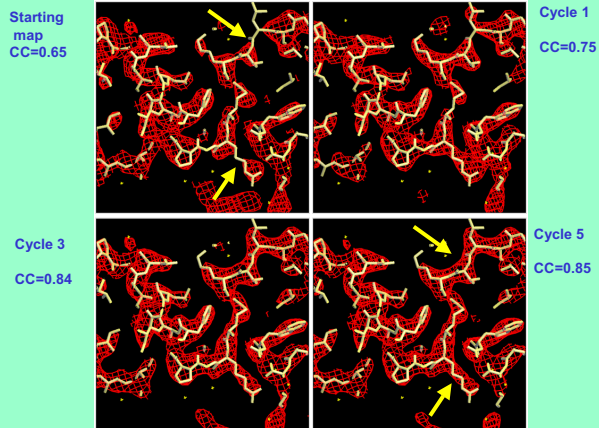


Image enhancement using local feature recognition (nusA protein structure)

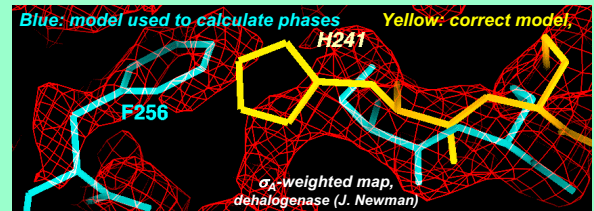


**Removing model bias with prime-and-switch phasing**

The problem:

Atomic model used to calculate phases -> map looks like the model

Best current solution:  $\sigma_A$ -weighted phases



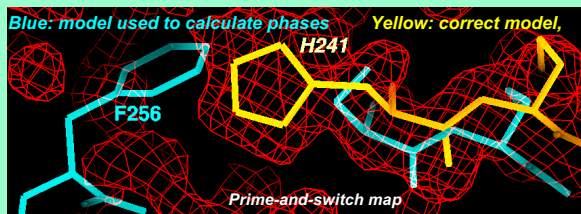
**Prime-and-switch phasing**

A solution:

Start with  $\sigma_A$ -weighted map

Identify solvent region (or other features of map)

Adjust the phases to maximize the probability of the map – **without biasing towards the model phases**

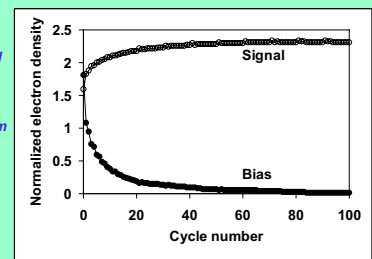


**Prime-and-switch phasing**

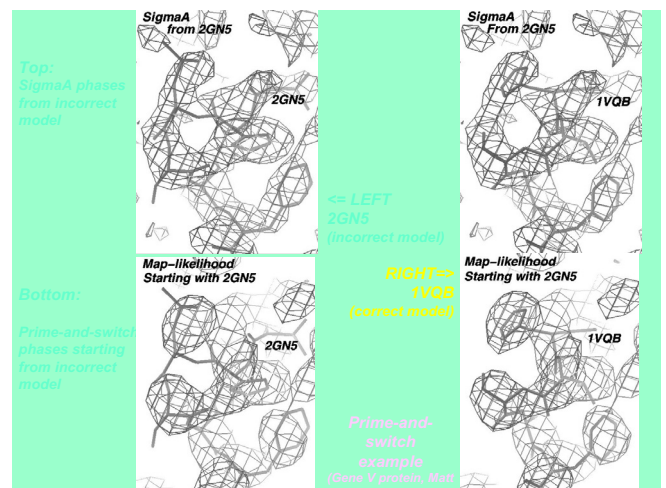
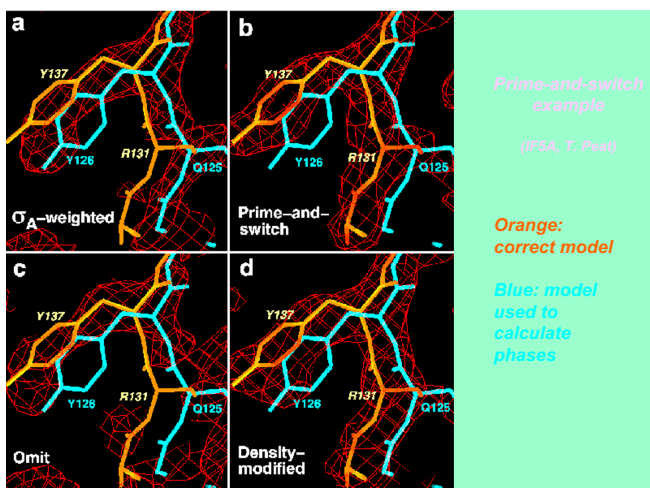
Why it should work...

**Priming:** Starting phases are close to correct ones...but have bias towards misplaced atoms

**Switching:** Map-probability phase information comes from a different source...which reinforces just the correct phase information

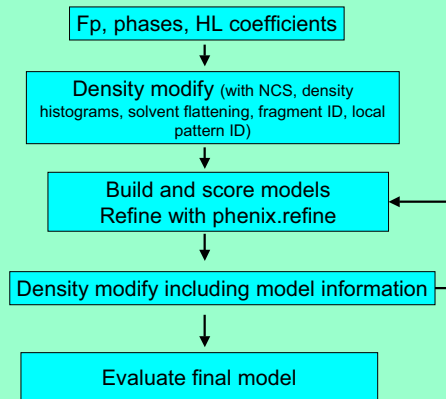


Signal: peak height at correct atomic positions  
Bias: peak height at incorrect atoms in starting model

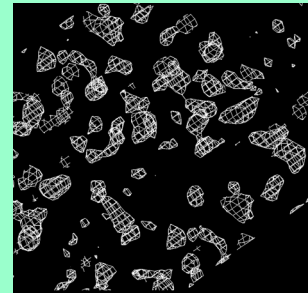




**PHENIX AutoBuild wizard standard sequence**  
(Following ideas from Lamzin & Perrakis)

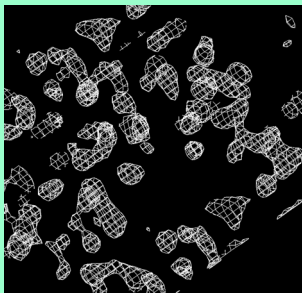


SAD data at 2.6 Å  
gene 5 protein



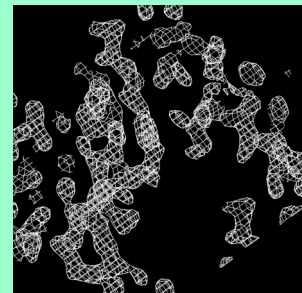
SOLVE SAD map

SAD data at 2.6 Å  
gene 5 protein



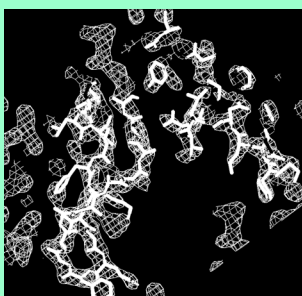
Density-modified SAD map

SAD data at 2.6 Å  
gene 5 protein



Cycle 50 of iterative model-building, density modification and refinement

SAD data at 2.6 Å  
gene 5 protein



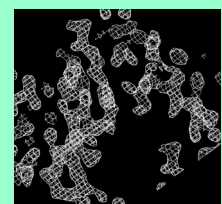
Cycle 50 of iterative model-building, density modification and refinement  
(with model built from this map)

Why iterative model building, density modification, and refinement can improve a map (following ideas of Perrakis & Lamzin):

1. New information is introduced: flat solvent, density distributions, stereochemically reasonable geometry and atomic shapes
2. Model rebuilding removes correlations of errors in atomic positions introduced by refinement
3. Improvement of density in one part of map improves density everywhere.



Density-modified map

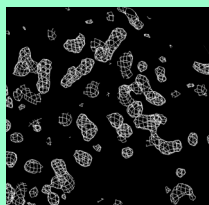


Iterative model-building map

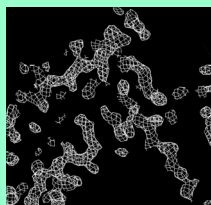
Iterative model-building and refinement is very powerful but isn't perfect...

Model-based information is introduced in exactly the same place that we will want to look for details of electron density

- How can we be sure that the density is not biased due to our model information?
- (Will density be higher just because we put an atom there?)
  - (Will solvent region be flatter than it really is because we flattened it?)
  - (Will we underestimate errors in electron density from a density-modified map?)
  - (Are we losing some types of information by requiring the map to match partially incorrect prior knowledge?)



Density-modified map



Iterative model-building map

A FULL-OMIT iterative-model-building map: everywhere improved, everywhere unbiased

→ Use prior knowledge about one part of a map to improve density in another

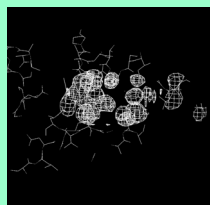
Related methods: "Omit map", "SA-composite omit map", density-modification OMIT methods, "Ping-pong refinement"

**Principal new feature:**

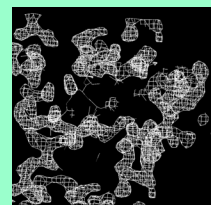
The benefits of iterative model-building are obtained yet the entire map is unbiased

**Requires:**

Statistical density modification so that separate probability distributions can be specified for omit regions (allow anything) and modified regions (apply prior knowledge)



OMIT region – no model, no NCS, solvent flattening optional

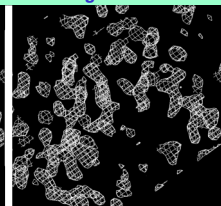
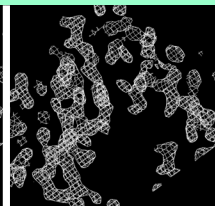
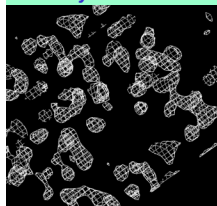


Outside OMIT region – full density modification

Including all regions in density modification comparison with FULL-OMIT

Density-modified

-----Iterative model-building-----



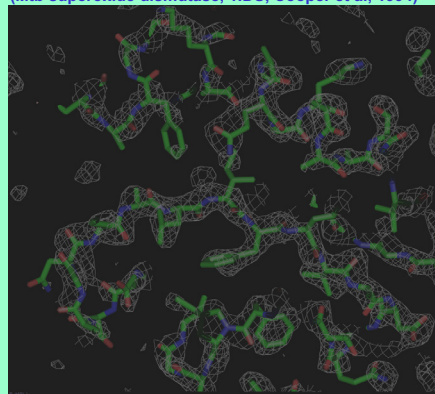
All included

FULL-OMIT

**FULL-OMIT iterative-model-building maps**

**Molecular Replacement:**

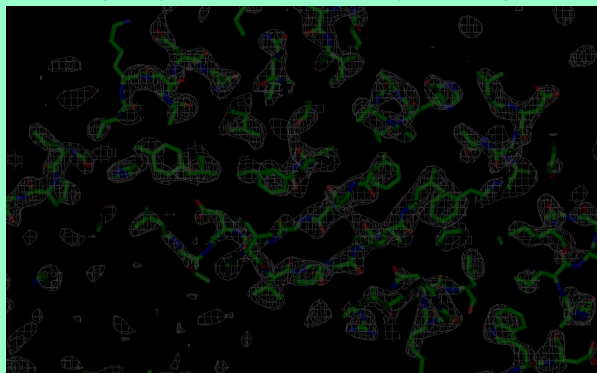
(Mtb superoxide dismutase, 1IDS, Cooper et al, 1994)



**FULL-OMIT iterative-model-building maps**

**Molecular Replacement:**

(Mtb superoxide dismutase, 1IDS, Cooper et al, 1994)



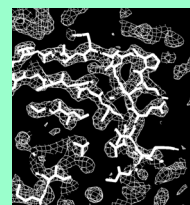
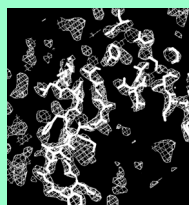
**FULL-OMIT iterative-model-building maps**

**Uses:**

Unbiased high-quality electron density from experimental phases  
High-quality molecular replacement maps with no model bias  
Model evaluation

**Computation required:**

~24 x the computation for standard iterative model-building



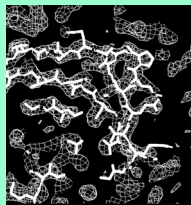
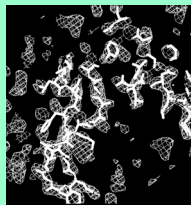


### FULL-OMIT iterative-model-building maps

Requirement for preventing bias:

Density information must have no long-range correlated errors  
(the position of one atom must not have been adjusted to compensate for errors in another)

→ Starting model (if MR) must be unrefined in this cell



### Major needs in automated structure solution

#### MAD/SAD/MIR

Robust structure determination procedures  
Best possible electron density maps to build most complete model

#### Molecular Replacement

Use of distant models  
Preventing model bias

#### All structures

Model completion/Ligand fitting  
Error analysis  
Decision-making for what data to use and what path to follow  
How to incorporate vast experience of crystallographic community

### Acknowledgements

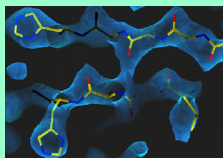
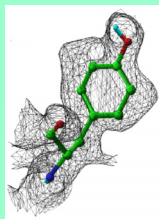
**PHENIX:** [www.phenix-online.org](http://www.phenix-online.org)

Computational Crystallography Initiative (LBNL):  
Paul Adams, Ralf Grosse-Kunstleve, Nigel Moriarty, Nick Sauter, Pavel Afonine, Peter Zwart

Randy Read, Airlie McCoy, Laurent Storoni,  
Hamsaprie (Cambridge)

Tom Ioerger, Jim Sacchettini, Kresna Gopal, Lalji Kanbi, Erik McKee, Tod Romo, Reetal Pai, Kevin Childs, Vinod Reddy (Texas A&M)

Li-wei Hung, Thiru Radhakannan (Los Alamos)



Generous support for PHENIX from the NIGMS Protein Structure Initiative

**PHENIX web site:**

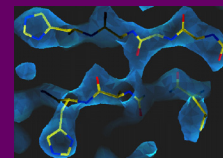
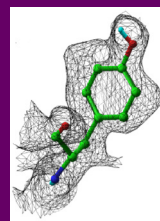
<http://phenixonline.org>

**SOLVE/RESOLVE web site:**

<http://solve.LANL.gov>

**SOLVE/RESOLVE user's group:**

[solve@LANL.gov](mailto:solve@LANL.gov)





Louis Farrugia

## Lecture-2

### Connecting programs together

#### Some crystallographic programs have complex functionality – *program systems*

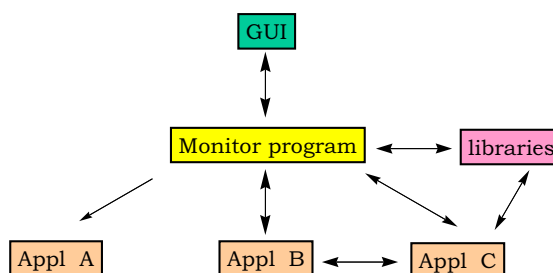
- PLATON, CRYSTALS
- DIRDIF, WinGX, GSAS, SHELXTL ...
- CCP4, XtalView, X-Plor, Phenix
- public SHELX suite
- monolithic program (single executable)
- separate executables connected together
- simply separate executables

#### Advantages of connected program systems

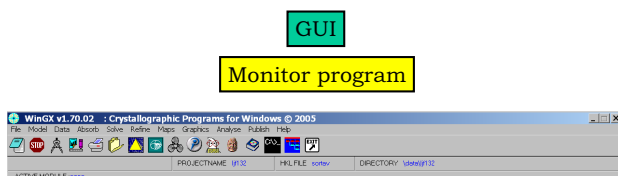
- file formats/data interconversions automatically handled
- monitor program(s) controls flow of processing by application programs
- changes easier to make - GUI/Monitor/applications can be updated separately
- coding errors *may* be less easily propagated - adding new code to monolithic programs may introduce unwanted side-effects

#### Disadvantages

- program interconnection *can* be dependent on operating system



#### Simple program system architecture



```
do
  if (menu1_selected) do X
  if (menu2_selected) do Y
  if (buttonA_pressed) do Z
  .
enddo
```

Event loop

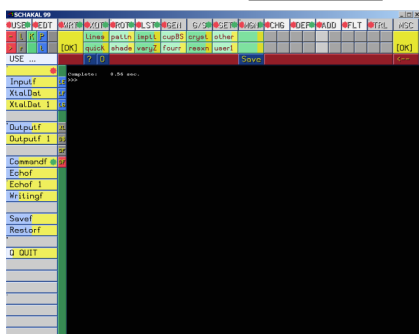
In WinGX system, executable wingx32.exe is both

#### GUI design

GUI design is more an art than a science

- keep it simple – limit the number of choices on an individual dialog box
- keep operations as standard as possible
- lay out controls neatly
- keep to the “look and feel” appropriate to your platform – use native windowing tools

## GUI design



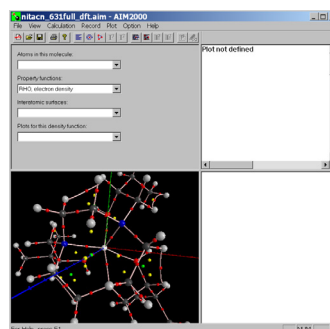
Symbols very cryptic

Need a very good understanding of the program to use GUI

Windows pop up with no obvious way of dismissing them

## GUI design

Don't include features which irritate users !



- pointless dialog boxes
- use all system resources
- no useful text output

The purpose of a well-designed GUI is to make it *easier* for the user.

## Libraries

All large program systems make use of libraries.

**library code** has general functionality and is used in several subprograms.

$a = \text{matmul}(b, c)$  –  $a, b, c$  are arrays

**library routines** have well-designed interfaces - reuseability is emphasised in OO programming.

**why use libraries ?** efficiency in programming – no need to reinvent the wheel

## Libraries

WinGX has 6 libraries implemented as DLL's

- [wgxlib00.dll](#) – mathematical functions – matrix inversion, eigenvalue, cell transformations, general routine, sorting, free-format parsing *etc*
- [wgxlib01.dll](#) – encapsulation of Salford routines
- [wgxlib02.dll](#) – PGPlot graphics libraries
- [wgxlib03.dll](#) – GETSPEC space group routines
- [wgxlib04.dll](#) – encapsulated Salford GUI routines
- [cifbtx26.dll](#) – CIFTbx version 2.6.2

## Libraries

Why use dynamically linked libraries ?  
( [<name>.dll](#) Windows [<name>.so](#) Linux)

- saves space – only one version needed – many executables can be linked simultaneously to same DLL
- makes correcting/updating large program systems easier
- code is linked at compile-time with library – changes can be made, but interface must remain the same

**Limitation** - must be self-contained set of routines - no calls to external routines.

## Libraries

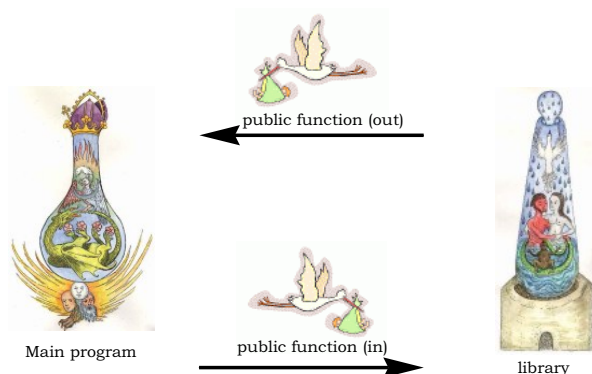
CIFTbx (modified version 2.6.2) implemented in library [cifbtx26.dll](#)

**Public functions passing information to DLL**

- [cifbtx\\_init\(\)](#) initialise all CIFTbx variables (initially in undefined state)
- [ocif\(<cif name>\)](#) load contents of existing CIF into CIFTBX memory
- [pcif\(<cif name>\)](#) creates new CIF in CIFTBX memory
- [dict\(<dic-name>\)](#) loads a CIF dictionary in CIFTBX memory for data validation
- [pchar\(<string>, <value>\)](#) puts value of CIF data item contained in <string> into new CIF

**Public functions returning information from DLL**

- [char\(<string>, <value>\)](#) gets value of CIF data item contained in <string>, in this case character value



## Libraries

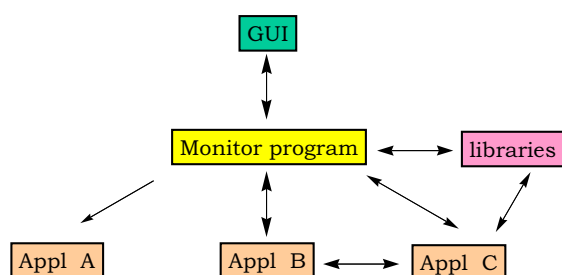
CIFtbx version 2.6.2 implemented in ciftbx26.dll

In WinGX, the library is used to :

- import CIF's and convert to SHELX files
- validate CIF's (IUCRVAL)
- write *archive.cif*, the summary file of structure determination

last functionality involves ...

1. concatenating all CIF's into one file (data\_ blocks)
2. reading request list of data items for *archive.cif*
3. sequentially finding all requested data items from concatenated CIF, and placing in *archive.cif*



**Simple program system architecture**

## Application programs

These are separate executables – capable of being run outside program system.

How does one executable program start another ?

Highly specific to compiler/language

- Unix Fortran – call system(<string>)
- Salford Fortran – start\_process@(<string1>,<string2>)
- C language – fork/exec

Scripting languages offer better solution for portability

Ousterhout's Tcl/tk scripting language is an example

## Application programs

How do programs communicate with one another ?

- files – the most portable method
- operating system specific messaging

Timing of events needs to be considered

*Monitor program*

```
.....
launch program(<programe>)
read results(<filename>)
.....
```

## Application programs

How do programs communicate with one another ?

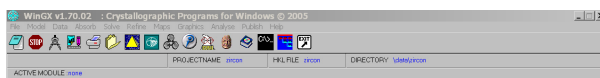
- files – the most portable method
- operating system specific messaging

Timing of events needs to be considered

*Monitor program*

```
.....
launch program(<programe>) – does control return?
read results(<filename>)
.....
```

### WinGX – a connected set of programs



Main program is WinGX32.exe – GUI + monitor program + file-handlers +....

/bin/ ~ 70 separate executables  
/files/ ~ 50 system files  
/manuals/ ~ 25 pdf files

Interfaces to external programs – SirWare, CCDC programs, POV-Ray, CRYSTALS, JANA2000

### WinGX – a connected set of programs



When WinGX main program is started ...

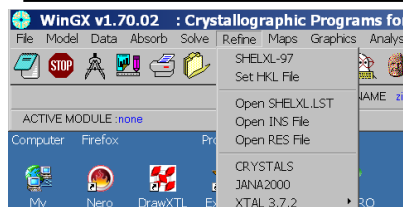
- initialises system variables and checks Windows state
- reads INI file -> current project name & location
- loads the current structural model into memory from SHELX files.
- executes code for GUI main Window, which enters event loop

### WinGX – a connected set of programs

```
call wxWindowCaptionW(MenuCaption)
call wxWindowSizeW(MenuWindowDepth,MenuWindowHeight)
call wxSetPositionW(0,0)
.
.
call wxAddMenuItemW(1,'&Refine[SHELXL-97]',1,fmenu6,0)
call wxAddMenuItemW(2,'Set HKL File',1,fmenu6,0)
.
.
call wxDefineClassNameW('WINGXMAIN')
call wxDefineMessageCallbackW(message_proc)
call wxCloseControlW(exit_proc)
.
call wxCloseDialogBoxW(0,i,0)
end
```

### WinGX – a connected set of programs

```
call wxAddMenuItemW(1,'&Refine[SHELXL-97]',1,fmenu6,0)
call wxAddMenuItemW(2,'Set HKL File',1,fmenu6,0)
call wxAddMenuItemW(2,'|,Open SHELXL.LST',1,fmenu6,0)
call wxAddMenuItemW(2,'Open INS File',1,fmenu6,0)
call wxAddMenuItemW(2,'Open RES File',1,fmenu6,0)
if(SysExec(14) /= ' ') then
    call wxAddMenuItemW(2,'|,CRYSTALS',1,fmenu6,0)
endif
```



## Program Suites

Harry Powell MRC-LMB

1 general introduction to suites

2 a quick look the differences between a small molecule/powder suite and one for proteins

3 a more detailed look at CCP4 and an example of using its built in tools

A definition of a program suite:

A set of complementary programs designed for use within a common domain

The programs may have a common "look and feel" or might be tied together through some external binding (e.g. a GUI or a similar command-line syntax).

Why contribute to a suite rather than develop your own packages?

- infrastructure pre-exists & is already tested
- someone else has already looked at porting
- often a large and established user base
- plenty of help in using underlying structure
- licensing should have been worked out
- someone else can take care of distribution (and, to some extent, users & their problems)

Why *not* contribute to a suite and develop your own packages instead?

- you may prefer another licence (GPL, LGPL, etc)
- it can be more restrictive & harder to develop in your own style
- your software may pre-exist and would need major changes to fit in with the suite
- you may want to maintain independence
- you may want to compete with the status quo

Macromolecular Crystallography

- CCP4
- Phenix

Chemical Crystallography

- Crystals
- Platon/System S
- WinGX
- CCP14

Big differences:

- MX suites tend to be looked after by multiple developers, CX by a single person (or a tiny group).
- MX suites tend to be better developed for Unix systems, CX for MS-Windows.

## CCP4 model vs CCP14 model

Both are principally funded by grants from UK Research Councils.

CCP4 is a single set of programs which use a common input/output & data storage model; the software is ported and curated by a funded group of people, and licences are sold to commercial companies. Core funding currently by 5-year grant from BBSRC.

CCP14 is a collection of software which uses many file formats for i/o; no attempt is made to bring the various component programs into a common model. Funding currently by a 5-year grant by EPSRC.

## (1) Chemical Crystallography

CCP14 - proposed in 1994, funded since 1995 - gathers together several complete small molecule and powder crystallography tools e.g.

Crystals, WinGX, ORTEX, Platon/System S, GSAS

and loads of software for individual tasks e.g.

SHELX\*, SIR\*, *etc, etc*

These are all developed independently and made available by their authors. Essentially there is a single member of staff who does not develop the software.

## (2) Protein Crystallography

CCP4 then and now

1979 - various groups in PX in the UK set up a collaboration to produce a common framework for the essential software for protein structure solution (SERC funded).

2005 - 17 funded developers (supported by BBSRC + income from sales) + 7 other main contributors/developers. Executive committee of 7 people. Contributions from other groups.

Software distribution then and now

1979 - Relatively easy to share software produced in an academic environment (employers were happier to allow it with few restrictions).

2005 - Can be much harder to release software to the wider world; even public sector employers put stringent requirements on licensing & release of software, patent offices seem happy to ignore prior art in issuing patents. This can lead to problems for software developers (google for "erroneous software patents") both in protecting their genuine innovations and in infringing spuriously granted patents.

CCP4 contains

- ~175 programs (mostly Fortran) which cover most aspects of macromolecular crystallography
  - diffraction image processing
  - reflection dataset analysis
  - structure solution (MIR, MR, Direct Methods...)
  - model building
  - structure refinement
  - analysis of results
  - validation
  - etc...

CCP4 contains

- comprehensive software libraries for use by applications -
  - Crystallographic functions, e.g.
    - symmetry
    - unit cell manipulations
    - FFT calculations (© Lynn ten Eyck)
  - general operations, e.g.
    - smart file opening/closing for many OS's
    - reading/writing reflection files
    - reading/writing coordinate files
    - keyword parsing



CCP4 runs on all commonly used platforms ( Linux, Mac OS X, Irix, Tru64, Solaris, MS-Windows...)

- the hard work of porting has already been done
- CCP4 staff can give help in porting your application
- contact *via* [ccp4@dl.ac.uk](mailto:ccp4@dl.ac.uk)

**Q** If there are already 175 applications how can I contribute?

**A(i)** Some functions aren't covered (recent additions include molecular graphics (ccp4mg) and model building (COOT), but there is still room for more).

**A(ii)** Who said they're perfect? Some programs are decades old, and your ideas may be an improvement

**A(iii)** Contribute to *part* of the project but not to the core, e.g. SHELXS, ARP/wARP & Phaser have ccp4i interfaces but are not part of CCP4.

In CCP4, reflections are stored in an MTZ file which has a binary (*i.e.* not ASCII) format. In practice, this does not present problems in reading from or writing to the file - you just need to use the appropriate CCP4 tools (or you could try Ralf Grosse-Kunstleve's alternatives).

"The MTZ reflection file format ... [was] renamed from LCF for three of its progenitors, Sandra McLaughlin, Howard Terry, and Jan Zelinka"

An MTZ file is more than just a reflection file; it is a container for a data structure with experimental information, *e.g.*

- unit cell information (dimensions & symmetry)
- reflection data (h,k,l,F, $\sigma$ (F),I, $\sigma$ (I)), coordinates on image, etc)
- for multiple crystals

Can be viewed and manipulated using standard CCP4 programs, *e.g.*

- `mtzdump` - for viewing the contents
- `mtz2various` - for writing to other programs' formats (e.g. SHELX, TNT, X-PLOR/CNS/CIF...)
- `mtzutils` - to change the information contained

Examples of code to manipulate MTZ files;

<http://www.ccp4.ac.uk/dev/templates/templates.php>

```
* Title:
.
* Base dataset:
  0 HKL_base
  HKL_base
  HKL_base
* Number of Datasets = 1
* Dataset ID, project/crystal/dataset names, cell dimensions, wavelength:
  2 sorted
  hg
  camillo
  58.4479 58.4479 156.0206 90.0000 90.0000 120.0000
  0.99970
* Number of Columns = 9
* Number of Reflections = 4925
* Missing value set to NaN in input mtz file
* HISTORY for current MTZ file :
  From SCALA: run at 12:04:08 on 3/ 7/05
  From SORTMTZ 3/ 7/2005 11:25:14 using keys: H K L M/ISYM BATCH
  From MOSFLM run on 3/ 7/05
* Column Labels :
  H K L IMEAN SIGIMEAN I(+) SIGI(-)
* Column Types :
  H H H J Q K M K M
* Associated datasets :
  0 0 0 2 2 2 2 2 2
```

```
* Cell Dimensions : (obsolete - use crystal cells)
58.4479 58.4479 156.0206 90.0000 90.0000 120.0000
* Resolution Range :
  0.00117 0.19363 ( 29.223 - 2.273 A )
* Sort Order :
  0 0 0 0 0 0
* Space group = 'H32' (number 155)
OVERALL FILE STATISTICS for resolution range 0.001 - 0.194
=====
Col Sort Min Max Num % Mean Mean Resolution Type Column
num order Missing complete abs. Low High
labe
1
1 ASC 0 22 0 100.00 10.6 10.6 29.22 2.27 H H
2 NONE 0 12 0 100.00 3.6 3.6 29.22 2.27 H K
3 NONE -68 68 0 100.00 1.2 25.9 29.22 2.27 H L
4 NONE -78.8 36533.5 0 100.00 820.07 820.28 29.22 2.27 J IMEAN
5 NONE 8.5 2066.7 0 100.00 62.64 62.64 29.22 2.27 Q SIGIMEAN
6 NONE -98.0 36399.4 0 100.00 775.22 776.09 29.22 2.27 K I(+)
7 NONE 0.0 3171.7 0 100.00 65.21 65.21 29.22 2.27 M SIGI(+)
8 NONE -85.1 36632.4 0 100.00 782.03 782.57 29.22 2.27 K I(-)
9 NONE 0.0 2724.5 0 100.00 65.29 65.29 29.22 2.27 M SIGI(-)
```

No. of reflections used in FILE STATISTICS 4925

#### LIST OF REFLECTIONS

```
=====
0 0 51 5487.92 424.86 0.00 0.00 5487.92 424.86
0 0 54 146.24 38.18 0.00 0.00 146.24 38.18
0 0 57 2737.31 209.20 0.00 0.00 2737.31 209.20
0 0 60 542.89 66.86 0.00 0.00 542.89 66.86
0 0 63 1148.36 129.10 0.00 0.00 1148.36 129.10
0 0 66 67.84 53.68 0.00 0.00 67.84 53.68
1 0 -68 1095.55 64.61 1095.55 64.61 0.00 0.00
1 0 -65 51.30 27.21 51.30 27.21 0.00 0.00
1 0 -62 489.80 46.13 489.80 46.13 0.00 0.00
1 0 -59 101.72 33.04 101.72 33.04 0.00 0.00
h k l IMEAN SIGIMEAN I(+) SIGI(+) I(-) SIGI(+)
```

Access the information with low level tools (written in C) for reading from MTZ structure with FORTRAN

e.g.

LROPEN - opens mtz file & does some checks  
 LRINFO - numbers of reflections, data columns, ranges  
 LRTITL - title from file  
 LRCELL - cell dimensions of 1st crystal in the data structure  
 LRRSOL - overall resolution limits  
 LRSYMI - symmetry information (names & numbers, point & space groups, etc)  
 LRCLOS - close MTZ file

etc., etc...

<http://www.ccp4.ac.uk/dist/html/ccplib.html>

contains details on low-level ccp4 routines which deal with (among many other things) potentially machine dependent behaviour, e.g.

CCPFYP - sets up environment & parses command line arguments

CCPDN - opens files, forces program to address error issues

CCPONL - tests to see if program is being run interactively

LITEND - determined endedness of current architecture

CCP4 example - FORTRAN opening an MTZ reflection file & printing the header

```
PROGRAM MTZOPEN
IMPLICIT NONE
INTEGER MTZIN,MTZPRT,MTZERR
C CCP4 initialisations
CALL CCPFYP
C MTZ-specific initialisations
CALL MTZINI
MTZIN = 1
CALL LROPEN (MTZIN, 'HKLIN', MTZPRT, MTZERR)
IF (MTZERR.EQ.-1) THEN
  CALL CCPERR(1, ' LROPEN no such file for HKLIN')
  STOP
ENDIF
CALL LRCLOS (MTZIN)
END
```

CCP4 example - C opening an MTZ reflection file & printing the header

```
#include "/Users/harry/ccp4-5.0.2/include/ccp4/cmtzlib.h"
int main(int argc, char **argv) {
  int *iprint;
  MTZ *file1=NULL;
  if (argc != 2) {
    puts("Usage: mtzopen <file>");
    exit(1);
  }
  file1 = MtzGet(argv[1],1); /* opens MTZ file and reads header */
  if (!file1) {
    printf("FATAL: failed to read file \"%s\"\n",argv[1]);
    exit(1);
  }
  MtzAssignHKLtoBase(file1); /* assigns base dataset */
  if (*iprint > 0) ccp4_lhprt(file1, *iprint); /* prints header */
  MtzFree(file1); /* closes MTZ file */
  printf("Done:   closed file \"%s\"\n",argv[1]);
}
```

Build Fortran example:

```
f77 -o fmtzopen mtzopen.f $CLIB/libccp4f.a $CLIB/libccp4g.a
```

run:

```
cmtzopen HKLIN <filename>
```

Build C example:

```
cc -o cmtzopen mtzopen.c $CLIB/libccp4g.a
```

run:

```
cmtzopen <filename>
```

CCP4 development environment (UNIX)

1. download from [www.ccp4.ac.uk](http://www.ccp4.ac.uk)
2. install by non-root user -
  - set up local environment by editing "ccp4.setup-inst" (for csh/tcsh) or "ccp4.setup-bash" and source-ing it to create a set of environment variables (logicals) defining the basic CCP4 directory structure, containing (among others);

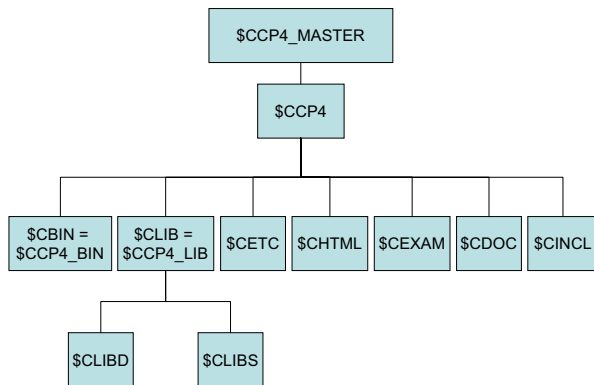
\$CCP4 - top-level directory

\$CLIB - directory containing link libraries

\$CPRG - executables

\$CINCL - run-time include files, e.g. for symmetry, extra logicals, ccp4.setup-#### files

#### Basic CCP4 directory hierarchy



Further online help -

<http://www.ccp4.ac.uk/dev/templates/templates.php>

basics for MTZ, maps, pdb files and symmetry lookup using CCP4.

<http://www.ccp4.ac.uk/dist/html/symlib.html>

general CCP4 programming tips:

<http://www.ccp4.ac.uk/dist/html/ccplib.html>

To summarize:

There are definite advantages to developing software within the context of a coherent suite such as CCP4 or PHENIX, since many of the issues regarding e.g. basic crystallography or file handling will have already been implemented as ready-to-use tools. This is at the cost of some personal programming flexibility, since someone else will have decided the rules!

## Outline

- Background
- Script languages for GUIs
- GUI design do's & don'ts

## GUI Design

Brian H. Toby  
NIST Center for Neutron Research

## Why use a GUI?

GUI = Graphical User Interface

- A well-designed GUI speeds learning
  - Opens software to occasional users & novices
- Scaleable: offers power tools to experts

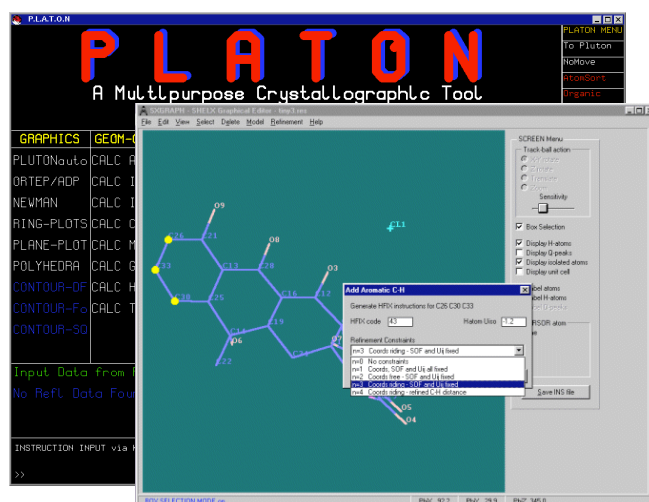
## Portable GUIs

Windows only?

Support for Linux & Mac offers wider range of users & growth into parallel processing

## Portable GUI tools

- Compiled (usually C++) packages
  - FLTK ([www.fltk.org](http://www.fltk.org))
  - wxWidgets [nee wxWindows] ([www.wxwidgets.org](http://www.wxwidgets.org))
- Virtual Machine
  - Java
- Script languages
  - Python + Tk, +wxWidgets, +GTK
    - GUI Builders: [wiki.python.org/moin/GuiProgramming](http://wiki.python.org/moin/GuiProgramming)
  - Tcl/Tk ([www.tcl.tk](http://www.tcl.tk) & [comp.lang.tcl](http://comp.lang.tcl))



## Pros & cons of scripting

### Pros

- Easy to code
- Test small routines
- Extensible when speed is needed
- Highly portable
- Add code at run time

### Cons

- Slower than compiled code
- Debugging can be non-trivial

## IMHO 1: GUIs do not need tremendous speed

- GUIs interact with people, who cannot tell the difference between a 10  $\mu$ sec vs a 50 millisec screen paint

## IMHO 2: Where possible don't incorporate code into script language, use external programs

When more extensive computations are needed, one can pass information to an external program, run it & read back results

- More portable
- Easier to debug
- More than fast enough: overhead of write, fork & read is usually trivial

## Example: Calling an external program

CMRPR EditCell replaced SGI GL program  
MANDEX: animate powder diffraction line positions

- 1st draft: run FORTRAN program each time slider is moved
  - Fast enough
- Final version: modify FORTRAN output for direct parsing by interpreter
  - Even faster!

## 1st vs. 2nd gen. output

ICD	H	K	L	MULT				
11	0	0	1	2	--	1	17.7176	5.00000
11	0	1	1	2	--	2	28.5434	3.12348
11	1							
11	1							
11	1							
11	-1							
11	0							
11	1							
11	-1							
11	0							
11	1							
11	-1							
11	0							
11	1							

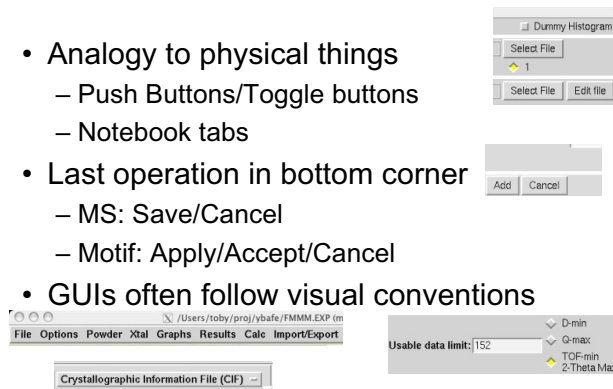
```

set dgenl(x) {
  17.7129707 22.191597 28.5358257 29.7368317
  34.8244629 35.8678513 37.4165802 41.6833496
  42.5831184 45.2750664 47.2591591 48.9819603
}
set dgenl(h) {
  0 0 1 1 -1 0 1 -1 0 1 -1 0
}
set dgenl(k) {
  0 1 0 0 0 0 1 1 1 0 0 2
}
set dgenl(l) {
  1 1 0 1 1 2 1 1 2 2 2 1
}
    
```

## Thoughts on GUI design

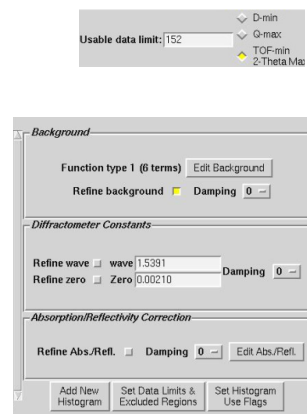
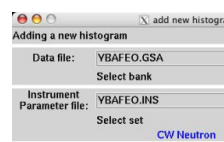
## 2nd Generation GUIs depend on a visual short-hand

- Analogy to physical things
  - Push Buttons/Toggle buttons
  - Notebook tabs
- Last operation in bottom corner
  - MS: Save/Cancel
  - Motif: Apply/Accept/Cancel
- GUIs often follow visual conventions



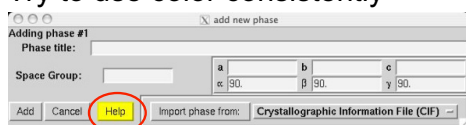
- Geographic proximity helps connect GUI components

- Separators (boxes, lines) keep sets of items distinct



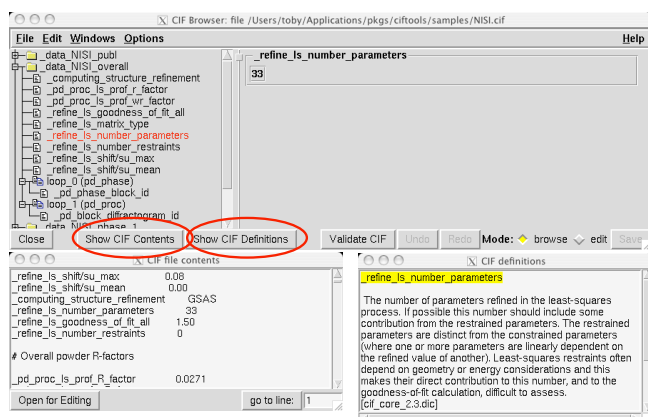
## Other Design Goals

- Screen space is valuable - don't waste it
- A little bit of color helps guide the eye
- Too much color is confusing
  - Keep contrast levels high (have pity on the color blind)
- Try to use color consistently

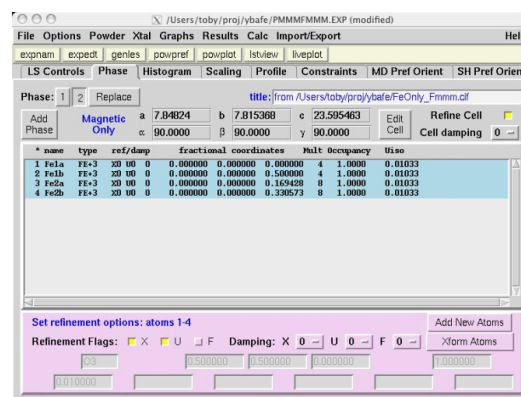


## GUI design: Hall of Fame & Shame

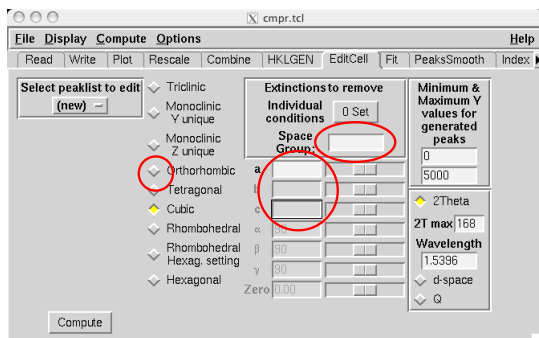
## CIFEDIT: not simple but easy



## Another of my “greatest hits” EXPGUI



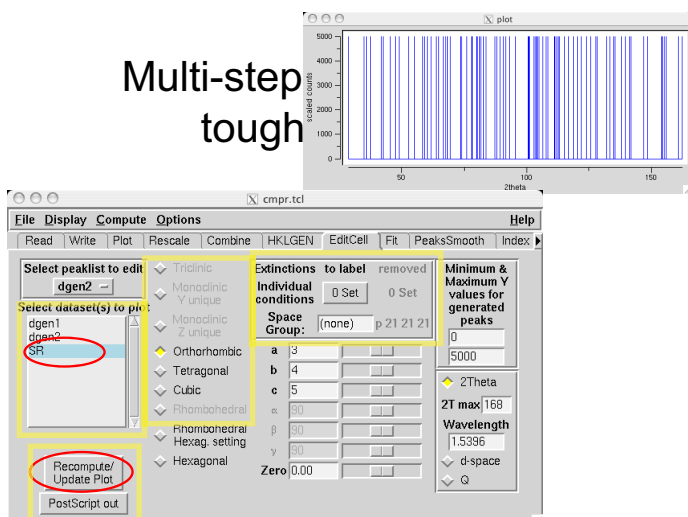
Multi-step processes are tough with GUIs



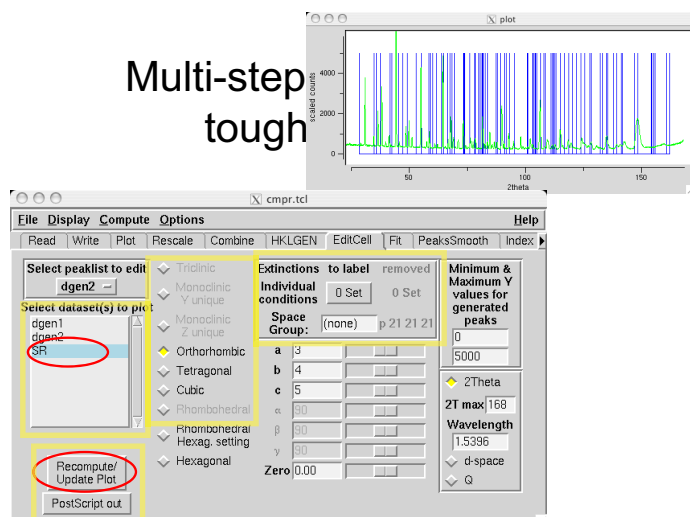
Multi-step tough



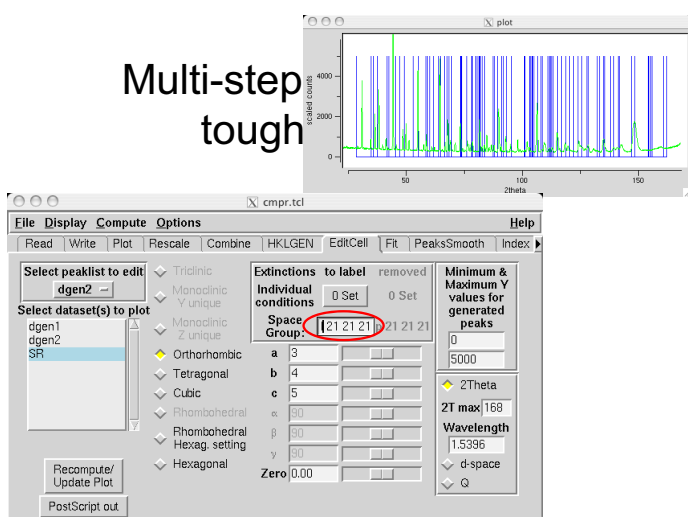
Multi-step tough



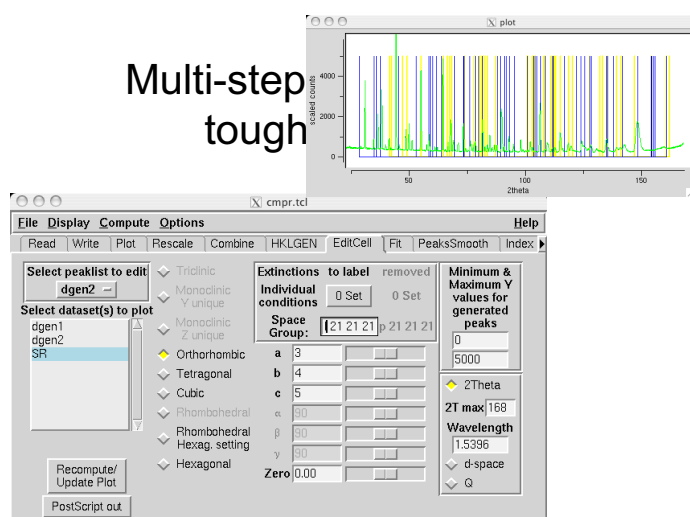
Multi-step tough



Multi-step tough

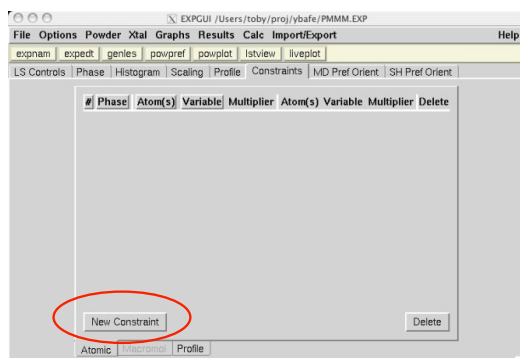


Multi-step tough

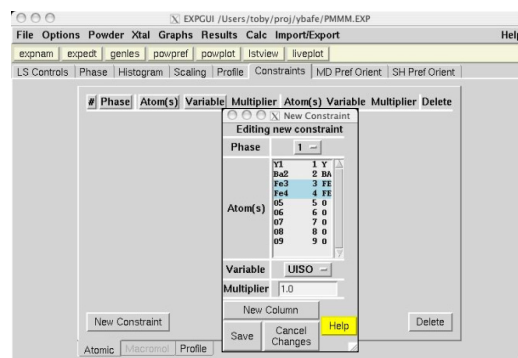




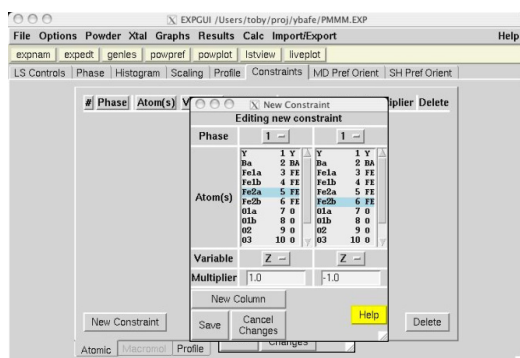
I don't know how to make this more intuitive



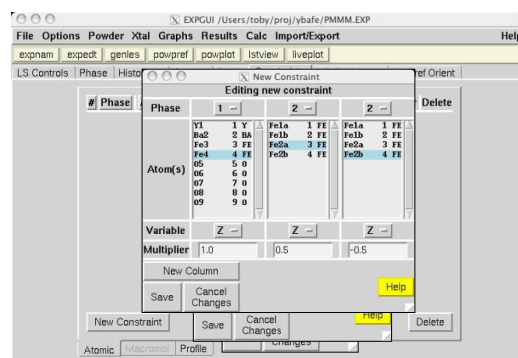
I don't know how to make this more intuitive



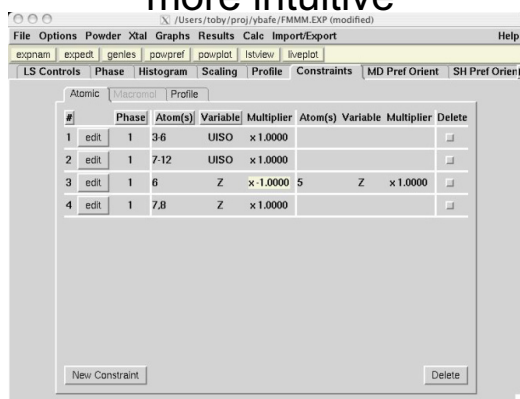
I don't know how to make this more intuitive



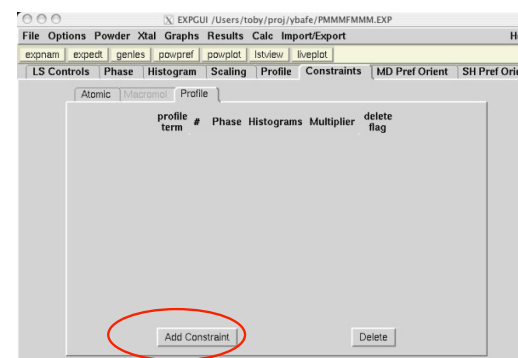
I don't know how to make this more intuitive



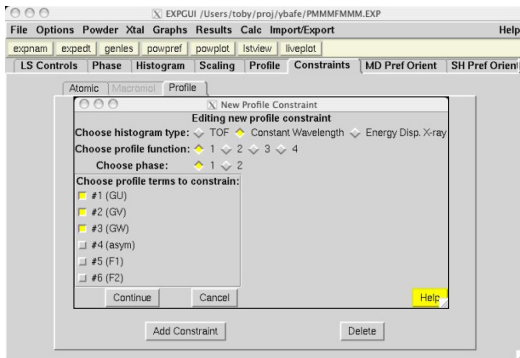
I don't know how to make this more intuitive



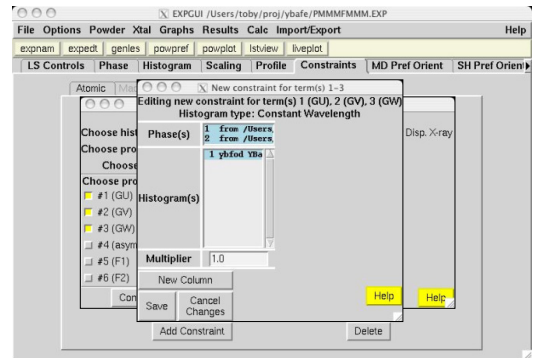
Even worse



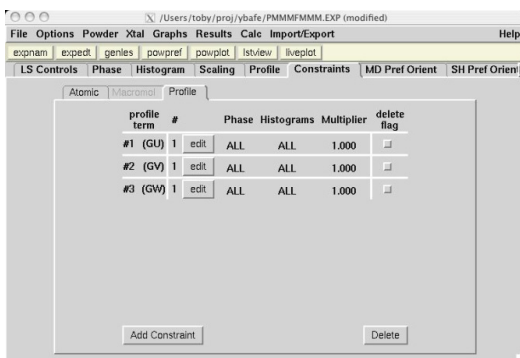
Even worse



Even worse

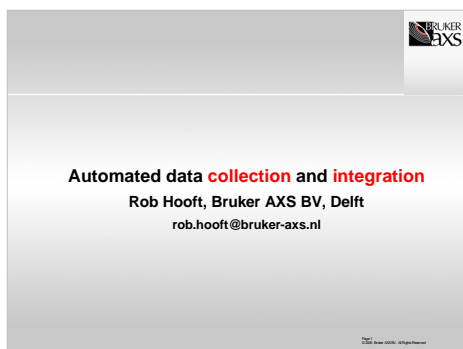


Even worse



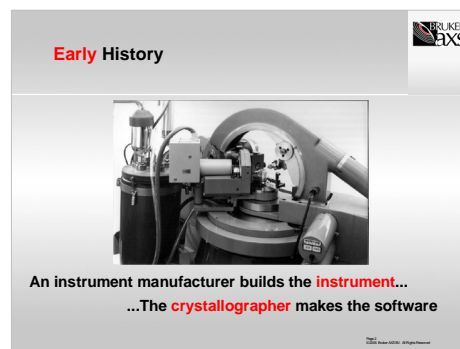
Conclusions

- Script languages are great for portable GUI design
- Intuitive GUIs take considerable thought
- Use conventional designs where possible
- Multi-step procedures are tough to make intuitive
  - Tutorials help
- Users really like GUIs



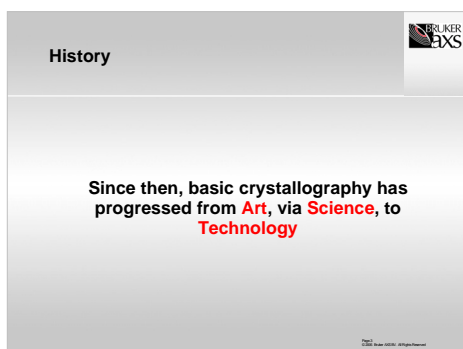
In this talk I will be jumping between IT background for crystallographers, and real crystallographic coding. I will show no direct crystallographic algorithms, but will focus on how they can be programmed in a future-directed way.

1

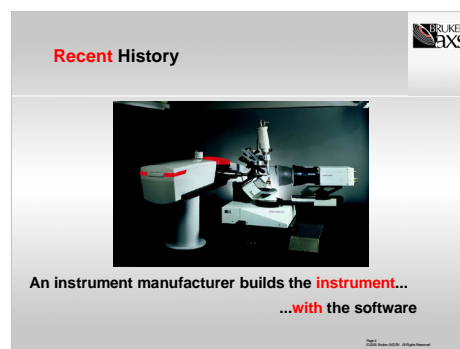


A long time ago, there was an instrument manufacturer which was basically a mechanics workshop. There was some firmware in the instruments, but all application software was written by the crystallographers that were using the equipment.

2



The instruments as shown in the previous slide were used when crystallography was mainly a goal. More and more crystallography is moving from goal towards a means, a method, to reach other goals like the explanation of a reaction mechanism for an enzyme. It is used as a technology.



More recently, the instruments have become a lot more complex, and a lot more accurate. The added complexity and accuracy requires a lot more instrument-specific programming. Also, because much of the application software is now no longer subject of the research being performed with the instrument, it is supplied by the manufacturer of the instrument.

**What software (for SCD) ?**

- Mount and center crystal
- Check whether it diffracts
- Determine a unit cell
- Check whether there is unexplained diffraction
- Make a data collection strategy
- Collect data
- Integrate data
- Correct data
- Solve structure
- Refine structure

More and more aspects of the software are integrated into the standard program tools, and more and more different crystallographic techniques progress from the research stage into something that can be routinely performed. Vertical as well as horizontally the software becomes more and more integrated. Structure determination of twins and incommensurate structures is now commonplace, integration of quasicrystals not yet.

**More Progress**

As more progress is made in the field, more and **more** applications are delivered with the instrument.

Roughly half of the development of a new instrument now is invested in software.

5

6

**Hardware dependence?**

A large fraction of the data collection software can be made **independent** of the exact instrument

For this to work, the easiest would be if there would be no assumptions about the hardware in the application software.

This is impossible: the software that once was used to integrate point detector data in small molecule SCD could not have foreseen the changeover to area detectors.

Many vendors have made a change from one-axis goniostats to 4-axis goniostats with accompanying changes in the code

But: if the design of the software is well thought out, it is possible to minimize the problems keeping the software running

**Motto of industrial software design**

You can not **predict** the future.

What will a machine look like in 5 years?  
What will it be able to do?  
Will the software be able to handle those changes?

If the future would be known, we could perfectly plan the software such that it could accommodate “future” changes. But even if we can not foresee the future it will be possible to structure the software such that it will accommodate changes in instrument design and experiment design with minimal efforts.

It is that structure that I would like to focus on in this class.

**Motto of industrial software design**

**You can not **predict** the future.**

**Instead you must try to **foresee** all possible  
different futures**

© 2002 IBM Corp. All rights reserved.

The lack of possibility to predict the future is a nice parallel between software design for crystallography and politics.

A recent paper in Scientific American that describes the importance of future developments in political decision making gave a nice alternative to predicting the future as the basis of a decision: predict a wide range of different futures, and make sure that the decision is performing well in as-wide a range of futures as possible.

This trick is valid as well for crystallographic software development.

## Application

So what does determine how adaptable we are towards future developments? To find out, we will have a look at some data from a diffraction machine.

9

10

# What are Data?



© 2002 by the Board of Regents of the University of Wisconsin System

This is a diffraction image from a CCD instrument.

Is this “data”?

# What are Data?

The image shows a screenshot of a Microsoft Excel spreadsheet titled "Plant Dump". The spreadsheet contains a table with 15 rows and 15 columns. The columns are labeled 204 through 226. The rows are labeled 204 through 226. The data is as follows:

	204	215	216	217	218	219	220	221	222	223	224	225	226
204	54	65	48	66	59	45	92	62	40	43	45	44	50
205	36	48	60	56	92	57	62	40	43	45	44	50	59
206	46	56	55	56	53	76	75	59	40	44	50	59	
207	37	52	53	65	106	168	156	109	73	61	63	42	
208	38	43	46	123	234	643	437	204	111	74	88	47	
209	42	52	92	216	715	1498	1126	341	122	63	56	52	
210	47	51	75	139	846	1703	1280	418	150	67	52	47	
211	45	55	65	136	462	956	816	203	107	57	63	58	
212	45	63	57	89	162	203	206	151	95	56	45	46	
213	37	41	56	83	91	96	38	88	64	54	48	46	
214	43	42	56	66	51	75	65	53	62	44	46	40	

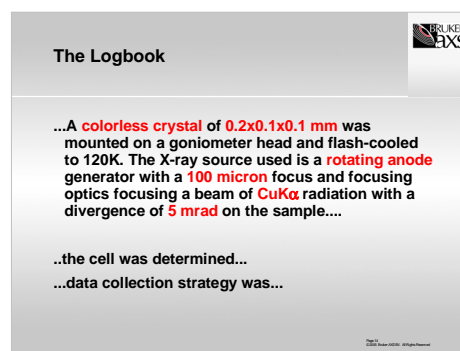
This is a part of the previously shown diffraction image, representing the exact numbers underlying the image. It represents the area around a single reflection.

- Can we integrate this reflection?
- How would it look in a HKL file?
- Which reflection is this?
- Is this “data”?



I indeed think that the answer to the question asked in the previous two images is “yes”: the diffraction image **is** the data. But the next question is “is data all we need”?

What we are missing is HOW this data was obtained. The logbook of the experimenter.



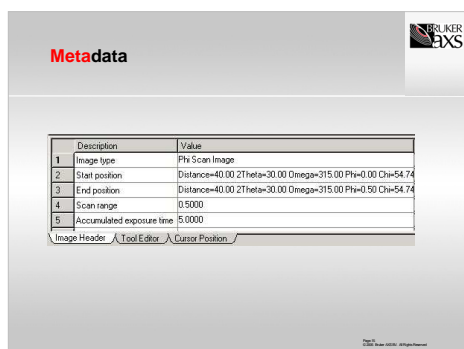
What I mean by the logbook is basically what all crystallographers write or at least used to write in an “experimental” section of a paper about how the structure determination was performed.

Given here are just a few examples.

This information is essential: not just to be able to repeat the experiment (*i.e.* for scientific reasons), but plain and easy because without additional information we do not know what reflections are visible on each diffraction image, and structure determination is impossible. All of the additional surrounding information should be present in a logbook.

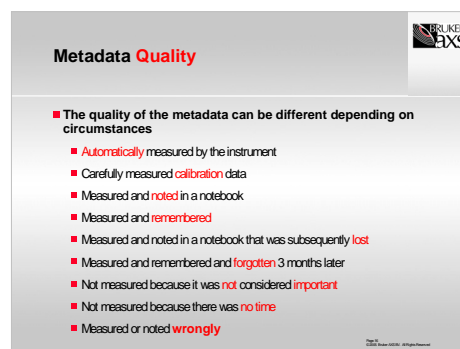
13

14



In information technology, the name for the information surrounding the data, the information that describes **what** data we have is *metadata*.

Not only the logbook of the experimenter, but also the logbook of the instrument itself provides metadata. For a 2D diffraction image, this information is often stored in the *image header*.

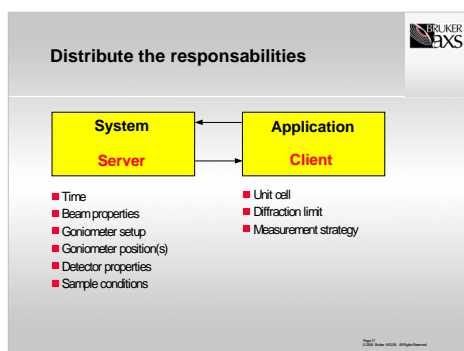


Metadata comes in different qualities. Some metadata is better than others. Lets go through a list of metadata from good through bad.

Even though the later items in this list may come across as humor please be assured that I mean this very seriously!

• Sometimes after performing experiments at a synchrotron people come back to the lab with a data set, but they do not remember the orientation of the phi axis of the goniostat.

• More involved: if the primary beam position is given somewhere in the metadata, this needs *metametadata* to be interpretable: x,y or y,x; mm or pixels, up/left or down/right as positive axes? And the values can be there from a previous determination three days ago...

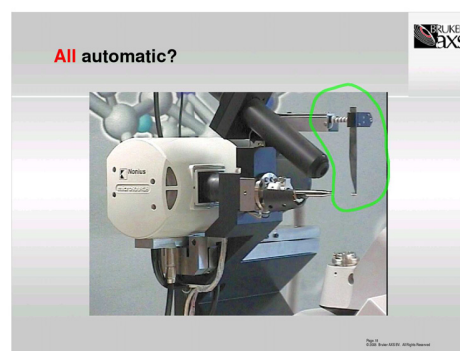


Another way to classify the metadata is to see who should be responsible for its determination. A diffraction experiment can be separated into two parts. In fact this is how we have written software where these two parts are written as a client connecting to a server program.

Responsibilities of the server include everything that is part of the instrument, and responsibility of the client is everything that is part of the experiment.

Not all is so clear:

- where is the crystal shape stored?
- The data-collection strategy depends on the hardware configuration.



For a number of years I have been involved in the development of the KappaCCD, an instrument that automated most of the motions and measurements of the system. The metadata could therefore be quite trivially recorded.

But not all of the instrument was wired: the beamstop, indicated here, could be manipulated by the user.

17

18

**Manual beam stop**

- Beam stop is **movable** and exchangeable. But: No sensors.
- Which beam stop and where it is positioned is entered into the program **by the user**.
- On the server side, the information is used for **collision avoidance**
- On the client side, the information is used to calculate which parts of the detector can not be trusted (**obscuration**).

Does every user always set these parameters?

The beamstop could be exchanged and moved by the user, but there was no sensor (except for a “beamstop present and in place”).

The user is supposed to note the change of beamstop identity and/or position to the program, but this is often forgotten.

If it is, the metadata registered in the image header regarding the beam stop is incorrect, and the unexpected position of the beam stop can cause an annoying collision with the detector.

**The best quality metadata**

The best quality metadata....  
....comes from the **most automated and integrated system**

An important conclusion can be drawn from this: The best logbook is one that has been kept by the instrument computer itself!

Metadata that has human involvement is by nature more unreliable than 100% instrument-determined metadata.

Of course, to have instrument-determined metadata one needs the instrument to measure as much as possible by itself. Many, but not all of the parameters stored as metadata are also suited for automation. As the MAR dtb system shows, a beam stop position can be motorized as well, and the size of the beam (the collimator is another thing that is not automated in a KappaCCD) can be determined by motorized slits.





## Prepared for the future

**How to be prepared?**

- We make sure that we have appropriate **metadata**.
- We keep the metadata at the appropriate location


**Is that sufficient?**

- We also have to make sure that the actual software will survive the years.

©2002 SLIKER AXS. All rights reserved.

OK. So, a necessary condition for making future-proof software is to keep appropriate meta-data, and to keep that meta-data at the appropriate location.

This, however, is not a guarantee for future success of our current software. We need to make sure that the software is written in a future-proof manner itself too.



## Software design

- As few assumptions about the environment as possible
- Run on different platforms
- Use a productive programming language
  - Performance is less important than you think
  - Do not spend any time optimizing rarely executed code
- Use the right programming techniques
  - Readable code
  - Portable code
  - Documented code
  - Object oriented/modular code
  - Group-development: code reviews, coding standards, sharing modules.

©2002 SLIKER AXS. All rights reserved.


Often, software written at a lab for internal usage contains, at least initially, a lot of hidden dependencies on the lab environment, like the location of other software package on the system. For future-proof software this is not a good idea.

Relying on a single hardware platform is also a bad idea. e.g. look at the history of DEC's VAX computers, and the history of bus architectures. Anything that was critically dependent on one of the dead technologies has died with it.

Optimize the time spent programming vs, the computer time saved: assembly language routines may be fast, but programmer time may be more expensive! These parts of the design are everyone's own responsibility. In the following part, we will discuss some examples of programming techniques.

21

22




## Intermezzo

- **Is hand-coded assembly really the fastest code?**
  - Is an assembly coder likely to use the fastest existing algorithm?
  - Did you ever use bubble-sort?
- **Is python code slow?**
  - Or is 99% of the execution time in a single loop that can be optimized?
  - Did you ever profile your software?

**Using existing libraries gains you speed and flexibility that was coded by experts.**

©2002 SLIKER AXS. All rights reserved.

Someone writing low-level code is likely to choose algorithms he knows, and spending large amounts of time optimizing the execution speed. Simply using another algorithm unknown to the user may gain an order of magnitude in speed.



## Example of future-proof coding (1)

**"Count the total number of 'a' characters in all descriptions in a dictionary that contain at least one 'z' character."**

```
def counta(dictionaryfile):
    totala = 0
    for line in dictionaryfile:
        if line.startswith('description'):
            if 'z' in line[12:]:
                totala += line[12:].count('a')
    return totala
```

©2002 SLIKER AXS. All rights reserved.

This is an example of a simple programming exercise that has been solved exactly as stated.

The subprogram is written in pseudo code, any resemblance to existing programming languages is purely coincidental.

The hidden assumption of the programmer is that none of the conditions set will ever change.

### Example of future-proof coding (2)

- This subroutine needs changes if:
  - The dictionary is no longer on a file
  - The dictionary is stored in a set of files
  - The dictionary file is no longer in a line-by-line format
  - The record structure of the dictionary changes
  - The character encoding of the dictionary changes from ASCII to UTF-16
  - The conditions placed on the counting are changed
  - Something else than counting would take place
  - We would need to count 'c' characters

A lot of imaginary future developments will require changes to this code.

In fact, the differenced can be grouped into five different groups:

- The location of the data
- The format of the data
- Data filtering
- Data analysis
- Parameters of the analysis

Does this suggest how the initial code example can be improved?

### We need to **split** responsibilities

- Four components:
  - Data source
  - Data decoder
  - Data filter
  - Data processor with parameters

Yes! The software should be split into independent components that work together to perform the given task. A change in the environment will then require only a change in one of the modules.

25

26

### Advantages of the **modular** approach

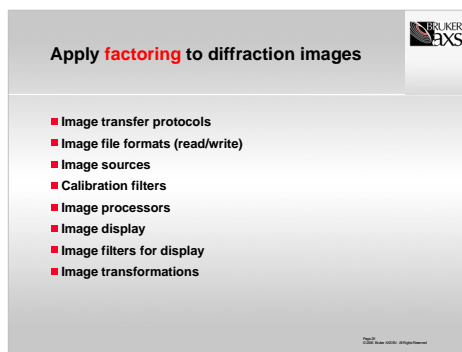
- Multiply instead of add up; re-use instead of copy code.
  - 4 sources, 4 decoders, 4 filters and 4 analyzers make 256 different combinations, not 16 (or 4)
- Algorithm abstraction
  - While writing a filter, it is not important to know the source of the data
- Parallel development
  - Once the interfaces have been defined, different people can work on the code in parallel without discussing the implementations all the time.
- Expert developers
  - Let a database manager write good database access code, and an dictionary specialist write good filters.

Several advantages are gained by this modular approach.

### Lets apply **factoring** to crystallography

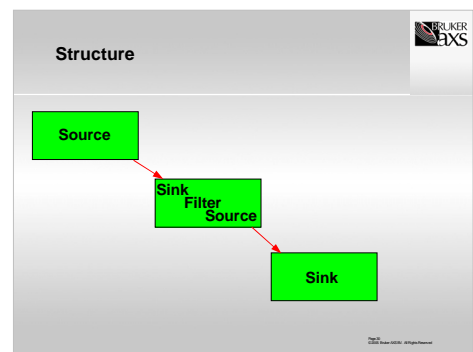
- goniometer positions
- goniometer motions
- other instrument operations
- low-level communication protocols
- high-level instrument protocol drivers
- crystallographic data format decoders/writers
- data collection strategies
- diffraction calculations and X-ray tracers
- ...

- Goniometer positions: get from and set to hardware, ask whether hardware can reach a position, convert between different types.
  - Operations on the hardware: scans, setting generator or cryostat. These can be combined into an experiment through a common application programmers interface.
  - Communication protocol at a low level: connection protocol to the hardware, high-level is command interpretation.
  - Data file formats: Program interpretation of different reflection file formats or diffraction image formats such that they can be used interchangeably.
- The actual implementation of each module is completely hidden behind a common interface. It is therefore possible to use external programs as well as internal implementations in a completely transparent fashion



We will use diffraction images as a more involved example of how we can factor code into different modules that can be made responsible for their own part of the work and to see what kind of gain we will have when we do that.

Listed here are a number of different tasks that can be performed on diffraction images.

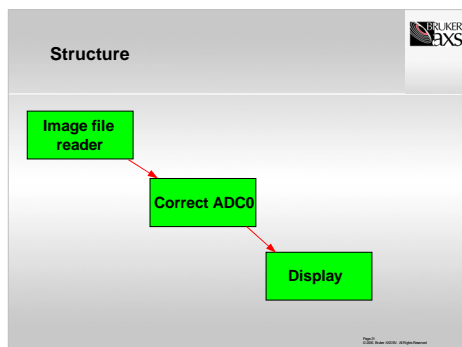


Now, instead of performing all operations that need to be performed on a diffraction image in a program in a simple succession of calls, we can see the program structured as modules that pass copies of the data and metadata around.

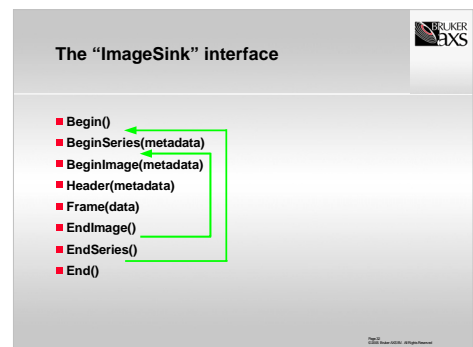
We can see three different kinds of modules on an abstract level: sources, filters and sinks. Since a filter can be seen as both a sink and a source, the protocol that is used for communication of an image from one to the other module (indicated with the arrows) is always the same

29

30

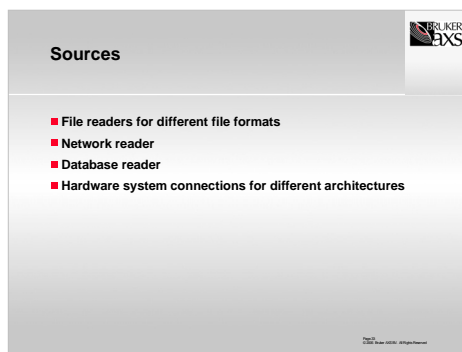


Here is a simple example of such a train of three.

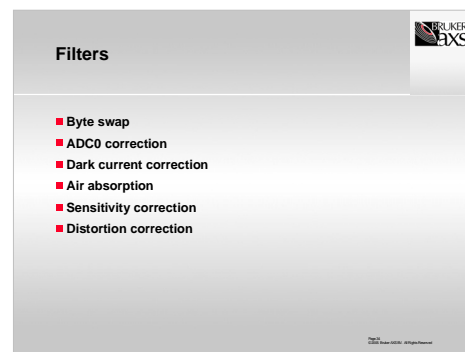


The communications protocol consists of a begin and an end message with zero or more series of zero or more diffraction images in between.

An *imagesink* is an object that can accept these messages (has these methods). An *imagesource* is an object that can emit these messages (call these methods).



Traditionally, application software always reads data from a diffraction image file. But if we have the flexibility in our software, we can think of several kinds of sources to be implemented. By the modularity magic, any program can use any source!



Many filters are traditionally implemented in a monolithic way. Separating each filter operation into its own module makes it possible to compose filters easily in ways that were not imagined before. One example here is the way a sensitivity correction is measured for a CCD detector. This is normally done with a flood-field image. Obviously, the calibration image should not be corrected for detector sensitivity, but it should be corrected for other effects like the dark current of the detector.

33

34

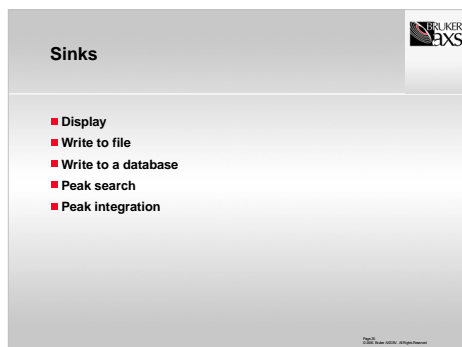
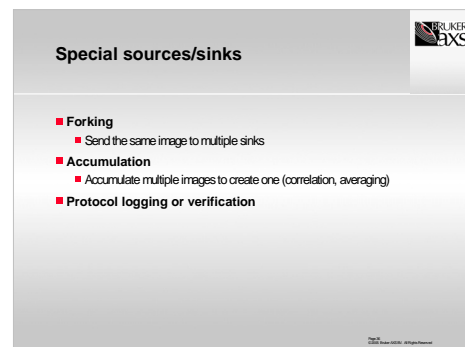


Image sinks are normally the operations that an application is written for. In the modular case the actual procedure will use object or module composition to build an application.

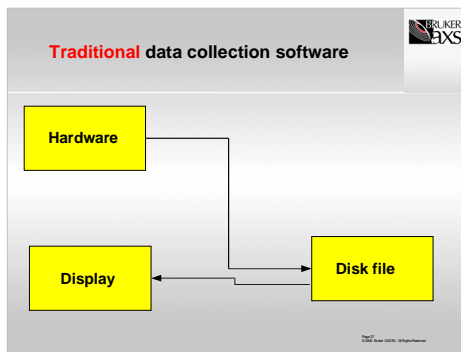
Note how easy it is now in theory to make a program that converts between any two different image file formats! One just needs to plug an image file reader source to an image file writer sink....

Different applications will use different compositions of sources and sinks. There is no forced code duplication anywhere.



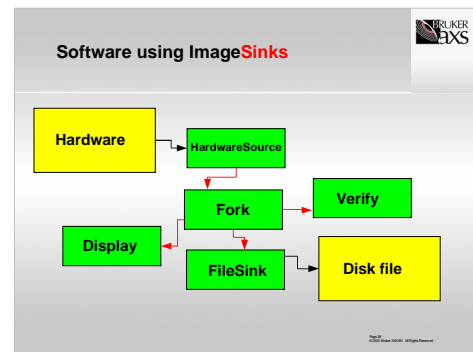
For house-keeping, we can make some special sources and sinks that make the composition of these objects even more flexible.

Protocol logging and verification sinks make it possible to make debugging the communication between sources and sinks a breeze.



What do we gain by the source/sink approach?

This is a simple diagram that shows how traditional data collection software works. There are two separate program threads: one is connecting to the hardware and is writing all the measured data and metadata to a disk file, and a second one is reading the disk file to display it on the fly.



The same software written with the image sink paradigm could look like this.

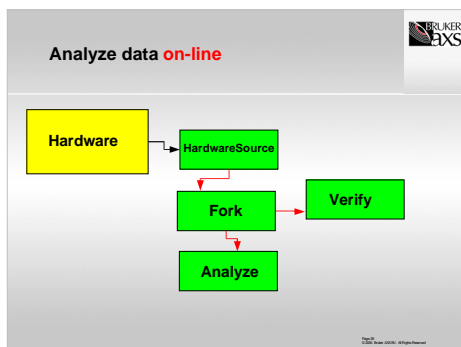
Red arrows are the image sink protocol.

Fork is a house-keeping module,

Verify is a debugging module.

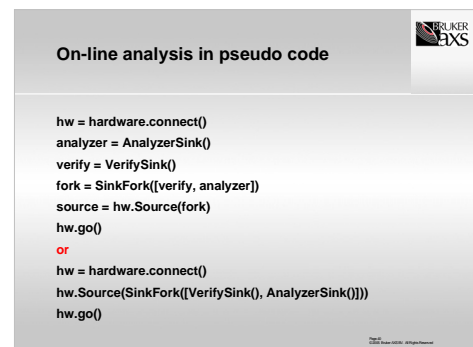
37

38



However, thanks to the imagesink protocol, it is now also possible to compose a completely different application that analyses data on-line without ever storing the information on disc.

Gluing the objects together in an object-oriented programming language is as simple as drawing the arrows in a powerpoint slide!



This is the diagram of the previous slide written in pseudo-code, both in a way that each of the objects stays available to the program, and in a more compact way that hides more of the internal structure of the sinks.

And now....

Now that we have all this beautiful **code**, we make it accessible through the **buttons** of a GUI

...or...

When is a GUI really handy? (1)

A GUI is really **handy** for operations that you perform manually **once a month**

An operation you perform three or **thirty times** in a row screams for a programming interface.

An operation you need to perform **automatically** can not have a GUI

41

42

When is a GUI really handy? (2)

A GUI is really **handy** if it can help you perform basic operations

A GUI that supports **every** possible operation that might be useful to someone someday is a **nightmare** for casual users

Typical examples of GUI's: word processor or spreadsheet.

Windows itself, with 500-page books with "Everything you ever wanted to know", and magazines full of "tips and tricks for using Windows" is a good example of why it is not advisable to use a GUI for every operation.

Scripting is good!

When is a GUI really handy? (3)

A GUI is really **handy** if the control flow is in the hands of the human controller

A GUI can be **annoying** in cases where the flow is decided by the program, such as in case of **automation**

If scripting does the work, the demand for a GUI becomes minimal. One will still need to look at the data, but that is not really a GUI.

The real power of a GUI program is unleashed when all operations of the program can be performed in any order, depending on what the user wants to do next.

**Use intelligent defaults:** Make sure that if the program has a good idea on what the user would want to do, that the default parameters are properly filled in!



## Extension language?


- We do not want a GUI that supports every possible operation.
- We do want to be able to perform the same operation many times in a row in an automated fashion.
- We want to be able to automate decisions and control flow.

Is the solution an **extension** language?  
Will we **write our own** extension language?

© 2002 Enraf-Nonius, All Rights Reserved

In an earlier stage we have seen that we need to be more careful with programmer time than with execution time.

Making an extension language has been very popular for a while (e.g. in the Enraf-Nonius CAD4 software) but it is not very efficient, and it is very difficult to write a good and complete extension language.



## A programming **library**!

- We choose a programming language
- We make a crystallographic library accessible from that language
- We write command line tools that use the library
- We write GUI's that use the library

Using a GUI designer from the beginning is a **dead end**!

© 2002 Enraf-Nonius, All Rights Reserved

There are very nice developer tools available now, where the development consists of


- Designing a GUI
- Adding subprograms under the buttons

Programs written in such a way are terminally attached to the GUI: it will never be possible to run the programs from a script, and extremely difficult to change the GUI toolkit at a later stage.

If a GUI developer is used, it should be made as a very thin layer over the top of existing code.

45

46




## A **users** point of view

- For normal **weekly** usage:
  - GUI access
- For routine daily or **hourly** usage:
  - Command line tools
- For advanced **special-purpose** usage:
  - Command line tools
- For **very advanced** usage:
  - Write a special application

© 2002 Enraf-Nonius, All Rights Reserved

Special applications can be used for routine operations as well if required.

Special applications could be either command line, or, if really needed, a GUI.



## Questions the **day after**....

- I know I changed something yesterday, what was it again?
- Why does it ask me to save? I didn't change anything?
- How come I can't reproduce these numbers now?
- Did I integrate all the data?
- Who the @\$% did touch that parameter?

© 2002 Enraf-Nonius, All Rights Reserved



And their **answers**...

■ Logging

■ Access control

■ Auditing

Formally

■ Good Laboratory Practice (GLP, GxP)

■ Keep a proper action log

■ A step further: CFR21/11

■ Requires logging in to the control software

■ Requires an unmodifiable audit trail

■ Requires digital signatures on all data

■ Requires that programmers know about CFR21/11

In its formality CFR21/11 adds annoyance to the process.

Pharmaceutical companies are bound to CFR21/11, especially in their process control, and a little less in their R&D.

49

50

Conclusions

■ Try to **predict** the future

■ Describe all things that **could** change in metadata

■ Avoid hidden dependencies and assumptions

■ Modularize, and **hide** implementations

■ Do **not** make **only** a GUI

Hidden assumptions are the goal of a programming exercise in the workshop.

# Operating Hardware

Rob Hooft, Bruker AXS BV, Delft  
rob.hooft@bruker-axs.nl

# Distribute the responsibilities

System  
Server

Application  
Client

- Time
- Beam properties
- Goniometer setup
- Goniometer position(s)
- Collision avoidance
- Detector properties
- Sample conditions

- Unit cell
- Diffraction limit
- Measurement strategy

We have seen this image earlier, there with the intention of making clear that the application should not deal with instrument parameters, but be able to deal with any set of relevant instrument parameters. Now we look at it from the other side: the server system will need to take care of all of the hardware-dependent issues of a crystallographic experiment.

1

2

# Distribute the responsibilities

System  
Server

Application  
Client

- Responsible for how the system is built
- Responsible for how the system can be used
- Responsible for system integration

- Responsible for how the experiment is performed

# Instrument

Goniostat

Cryostat

Generator

Sample robot

Shutters

Monochromator

Detector

Instrument

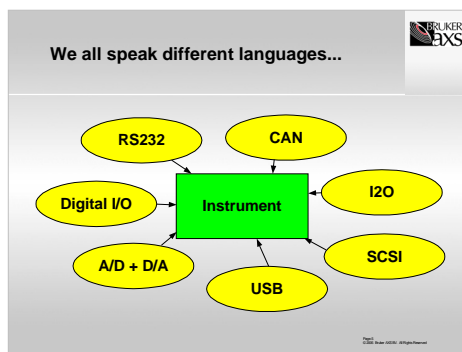
A crystallographic instrument consists of many parts that should collaborate. Having all of it integrated into one instrument is not strictly necessary, but it does allow experiments that would otherwise be absolutely impossible.

The minimum integrated system consists of a shutter, a goniostat and a detector (if the detector has an electronic shutter, the shutter can even be left out, but CCD detectors do not have that possibility).

Integrating the cryostat will allow one to do measurements at different temperatures in an automated fashion, and also to find out afterwards at what actual temperature a measurement was performed.

Integrating the generator will allow one to do measurements at different generator settings in an automated fashion, and also to find out afterwards at what actual settings an experiment was performed.

Integrating a sample robot will allow one to do measurements on a series of samples in an automated fashion, and also to find out afterwards for which sample an experiment was performed.



The big challenge for the integration of all of the parts into a properly configured instrument is that all the protocols differ, and different systems use other protocols. Tomorrows shutter may not use the same protocol as the one used today.

It becomes even more complicated if some components share one (RS232) connection like a generator that must be addressed via a goniostat, or a shutter that must be controlled via a generator.

Differences between crystallographic machines are as big as those between a DVD player and a toaster...

5

6

The crystallographic application software that controls the experiment should not know about these differences

**Drivers**

- Independent drivers
  - No changes needed in modules that control an unchanged component
- Use driver chains to uncouple responsibilities
  - Communication protocol and command protocol are often separate
- Multiple drivers for each component
  - Determine an "abstract" component interface
- Implementation hiding

Software engineers are often consulted only after the hardware has been built or bought.

Responsibilities of the server

- Time and timing
- Beam properties
- Goniometer setup
- Goniometer position(s)
- Collision avoidance
- Detector properties
- Sample conditions

We have already listed some responsibilities of the server software. We will now go briefly into each of the listed issues.

Time and Timing issues

The diagram illustrates the timing sequence for a single crystal diffraction experiment. It shows two variables over time: Motor speed and Angular position. The Motor speed starts at a low value, ramps up to a constant speed, and then ramps down. The Angular position starts at a low value and increases linearly during the constant speed phase. Key events are marked with vertical red lines: 'Detector enable' occurs before the motor starts; 'Shutter open' occurs at the start of the constant speed phase; 'Shutter close' occurs at the end of the constant speed phase; and 'Detector readout' occurs after the motor has stopped.

Timing issues are critical to the accuracy of the measurement. In a typical single crystal diffraction experiment the crystal is rotated with a constant speed during the measurement. Because the motors can not accelerate infinitely fast they are actually spun back from the starting position to ramp up their speed, and when they reach the starting position at the programmed rotation speed, the shutter will be opened. When the end position of the rotation has been reached, the shutter will be closed, the motors will be ramped down, and the CCD detector will be read out.

Beam properties

A photograph of an X-ray diffractometer setup, showing the X-ray tube, goniometer, and detector.

The properties of the X-ray beam depend on the X-ray tube (RAG/Sealed, type), optic (monochromator/multilayer optic), and collimator holes.

The application program should not be concerned by the exact hardware; instead the beam must be described in the form of some reusable parameters that can be used to describe any future and past beam

- Wavelength
- Polarization
- Convergence/Divergence
- Slits
- Pinholes

Goniometer setup

- Cascade of axes
- Alignment issues
- Zero-point calibration
- Maximum/minimum reliable speed
- Maximal acceleration

- Which axes, and connected how
- Alignment
- How fast/slow can the axes move
- How accurate is each axis

## Goniometer positions

- Conversions between different formalisms
- Where are you, where can you go
- Virtual goniostats: rotate around non-existing axes

If any conversion is done by the system, this may mean problems later.

Simple example: a rotation from  $\phi=0$  to  $\phi=90$ , was that a positive rotation around 90 degrees? or 450 degrees? or -270 degrees?

Interpolation is a different issue:

- Kappa  $\leftrightarrow$  Chi conversion
- Virtual goniostat

We assume that a rotation is linear in speed in the given axes!

Where are you, where can you go: -140 to -90 and +60 to +140 in omega. So: we can not scan from -130 to +130!

## Collision avoidance

- Where can we go
- How do we get there: efficient, safe
- Margins
  - Inaccuracies in description and in the mechanics
  - High-speed scans
- Escape from a collision
- Different sample stages
- Objects in the way

The more complicated the system, the more areas there are that can not actually be reached.

To calculate where one can go and where not is a very simple calculation in principle, it just requires an accurate description of how the machine looks. The accuracy is limited by mechanical tolerances and by how much time one wants to spend calculating.

It is much more difficult to calculate an efficient route from one to another point if not all points on the intermediate straight track are accessible.

13

14

## Detector properties

- Dark current
- Bad pixels
- Distortion and alignment
- Sensitivity
- Accuracy
- Point spread


Some **corrections** applied at the application level?

## Distortion correction

- pixel coordinate system is 2-dimensional
- mm coordinate system is 3-dimensional
- Use of other coordinate systems does not make sense.
- Distortion and alignment of the detector are "inseparable"
- Transformation for the distortion uses
  - spline
  - Normalized polynomials
  - Chebyshev polynomials

We had a talk about crystallographic coordinate systems, but the instrument has coordinate systems as well. The goniometer position is one, another is the detector that can be seen as an array of pixels, or as an array of 3D diffracted rays.

The transformation between these two consists of distortion correction and alignment of the detector.



## Application writer's **dream**

- The detector should be ideal:
  - Perfectly aligned
  - No point spread
  - No distortion
  - Same sensitivity everywhere
  - No dark current
  - Counting statistics
  - Not sensitive for anything but the X-rays from the sample.


Based on the image of an imperfect detector, one can correct for all effects in order. But is that really the way to go? Some of the corrections will really deteriorate the data: e.g. correcting for image distortion will make the pixel size larger than it really is (adding noise!)

Counting statistics is lost by corrections.

Furthermore it is possible during the integration to be “intelligent” about what is happening. Correcting for an unreliable pixel inside a peak can use a different algorithm than in the background.

Also, one should make an important distinction between how the image looks to the naked eye, and the quality of the integrated data. The eye is sensitive (and extremely sensitive) only to nuances in the background, and not at all to peak height!

17



## Sample conditioning

- Temperature
- Humidity
- Pressure
- Magnetic field
- ...

Sample conditioning may look a trivial requirement for most people, as the instrument may be operated at the same condition 99% of the time, but integration will give a proper reporting in the metadata, and is very useful in an inaccessible synchrotron system. It also allows for automation.


18



**Not all responsibilities are easily appointed to the **instrument** or to the **application**!**

We discussed a lot of the responsibilities of the server, but it is not always obvious where to place features.

An example are human interpretations of crystal looks. Another is the data collection strategy that depends on the detail of the instrumental buildup.



## Conclusions

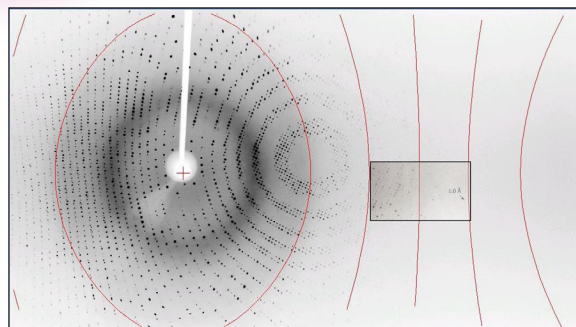
- Hardware development is as unpredictable as application requirements
- Keep a clear eye on whether the instrument or the application is responsible for an issue
- Modularizing allows for expert written code and maintainability

# Integration of 2D diffraction images

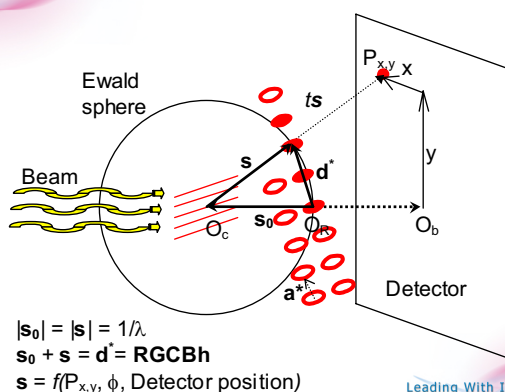
James W. Pflugrath<sup>1</sup>  
Rigaku/MSD, Inc., The Woodlands,  
Texas, USA

Leading With Innovation

## A diffraction image



Leading With Innovation



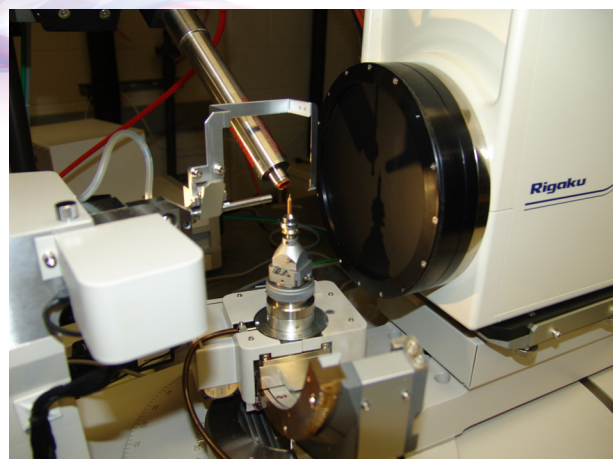
Leading With Innovation

## Diffraction math

$$s_0 + s = d^* = \text{RGCbH}$$

- h** Miller index ( $h, k, l$ )
- B** Crystal orth. matrix ( $a, b, c, \alpha, \beta, \gamma$ )
- C** Crystal orientation matrix
- G** Crystal goniometer matrix
- R** Rotation axis matrix
- d\*** Reciprocal lattice vector
- s<sub>0</sub>** Direct beam wavevector
- s** Scattered beam wavevector

Leading With Innovation



## What you do

- Pick up crystal in loop, plunge into LN<sub>2</sub>
- Put crystal on magnet on goniometer head and optical align
- Take a diffraction image or two
- Look at image(s) and decide whether to proceed
- Collect images, index, integrate, scale

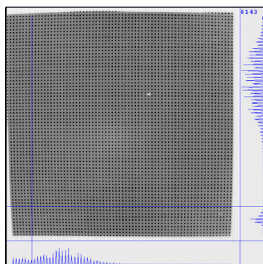
Leading With Innovation



## Detector - Calibration

Stanton, et al. (1992) *J. Appl. Cryst.* **25**, 549-558.

Dark current  
Non-uniformity of response  
Spatial distortion  
Bad pixels  
Zingers



Leading With Innovation

## Reflections (from images)

- Find
  - X, Y,  $\phi$
- Index
  - Unit cell
  - Orientation
- Refine
  - Crystal
  - Detector
  - Source
- Predict / Strategy
  - Rot start, end
  - Completeness
- Integrate
  - $hkl$ , Intensity,  $\sigma_1$
  - Profile fitting
- Scale
  - Rmerge
  - $|\chi^2|$

Leading With Innovation

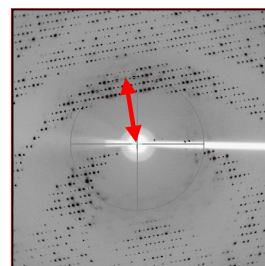
## Integrate

- Predict reflection position
- Put box around reflection
- Assign pixels to Peak and Background
- Sum Peak, subtract Background
- Profile-fit
- Apply correction factors

Leading With Innovation

## Direct beam position

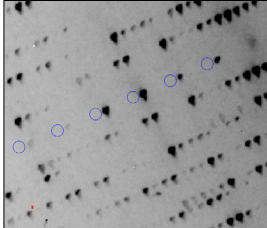
- Direct beam shot
- Powder rings
- Ice rings
- Symmetry



Leading With Innovation

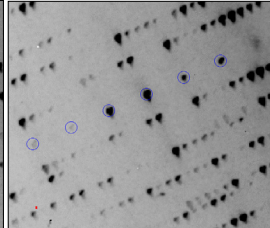
## dtdisplay overlay

Detail of 454 Angstrom axis



Beam center off (00l off by one)

Systematic absences and 2-fold



Beam center correct

Leading With Innovation

## Refine

- $\mathbf{s}_o + \mathbf{s}_i = \mathbf{RGCb}h = \mathbf{d}_i^*$
- $\text{Min } \chi^2 = \sum w_i (\mathbf{s}_{i,\text{obs}} - \mathbf{s}_{i,\text{calc}})^2$   
 $= \sum w_i [\mathbf{s}_{i,\text{obs}} - (\mathbf{RGCb}h - \mathbf{s}_o)]^2$
- Crystal (**B**, **C**): a, b, c,  $\alpha$ ,  $\beta$ ,  $\gamma$ , Rot1, Rot2, Rot3
- Detector ( $\mathbf{s}_{i,\text{obs}} = f(\text{Det}, X, Y, \phi, \mathbf{R})$ )
  - Beam center, Distance, Rotations ( $2\theta$ )
- Source ( $\mathbf{s}_o$ ): Direction, wavelength

Leading With Innovation

## Integrate

- Predict reflection position
- Put box around reflection
- Assign pixels to Peak and Background
- Sum Peak, subtract Background
- Profile-fit
- Apply correction factors

## Integrate - Predict

$$\mathbf{d}^* = \mathbf{x}_r = \mathbf{RGCb}h \quad \mathbf{x}_r - \mathbf{s}_0 = \mathbf{s} \rightarrow \mathbf{P}(\mathbf{x}, \mathbf{y})$$

Rocking curve

$$R_r = 2L[\Delta d^* \cos \theta + (\delta \lambda / \lambda) d^* \sin \theta]$$

Lorentz factor

$$L = |1.0 / (\mathbf{x}_r \cdot (\mathbf{r}_1 \times \mathbf{s}_0))|$$

Polarization factor

$$p = 1 - [P_n * (\mathbf{s} \cdot \mathbf{s}_n)^2 + (1 - P_n) * (\mathbf{s} \cdot \mathbf{n}_p)^2]$$

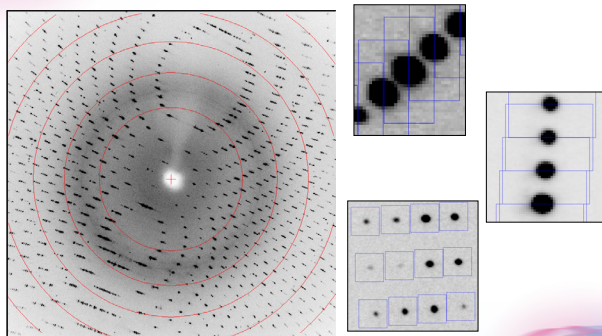
Oblique incidence correction factors

$$O_1 = (1 - \exp(f_{obl})) / (1 - \exp(f_{obl} / \cos \alpha))$$

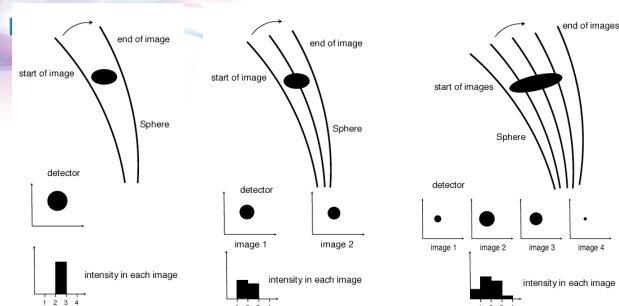
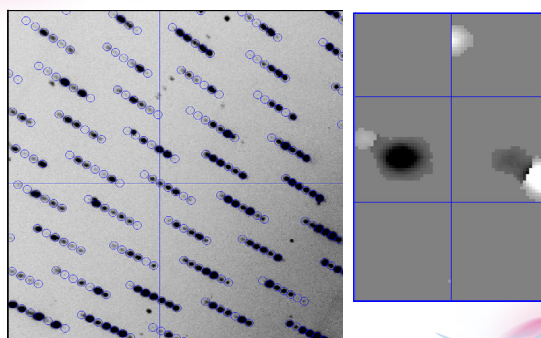
$$O_2 = \exp(f_{obl}) / \exp(f_{obl} / \cos \alpha)$$

Kabsch (1988) *J. Appl. Cryst.* **21**, 916-924.Zaleski, Wu & Coppens (1998) *J. Appl. Cryst.* **31**, 302-304.

## Integrate - Put box around



## Integrate - Put box around

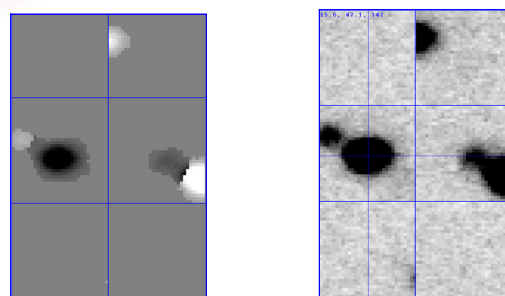


A fully-recorded spot is entirely recorded on one image

Partials are recorded on two or more images

"Fine-sliced" data has spots sampled in 3-dimensions  
Perhaps best processed with a 3D program (eg d\*TREK, XDS)

## Integrate - Background

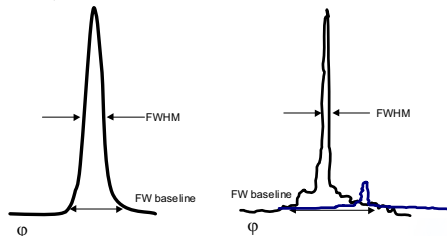


Background: Least-squares fit to a plane

## Refine (crystal mosaicity)

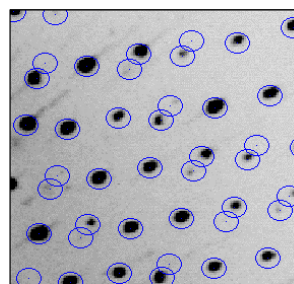
Rocking curve

$$R_i = 2L[\Delta d \cdot \cos\theta + (\delta\lambda/\lambda)d \cdot \sin\theta]$$

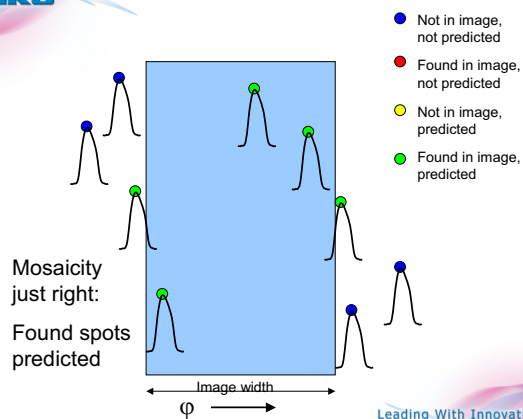


Leading With Innovation

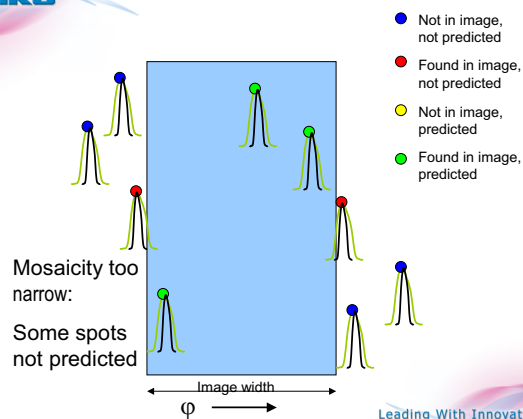
## Refine (crystal mosaicity)



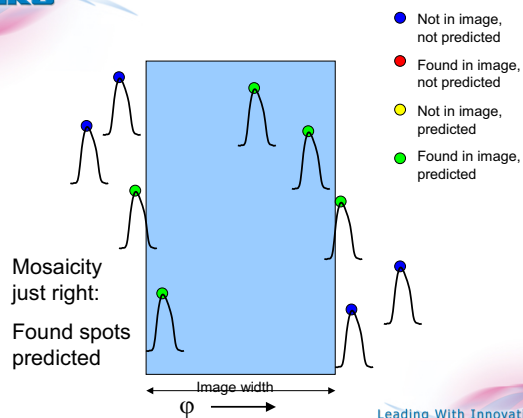
Leading With Innovation



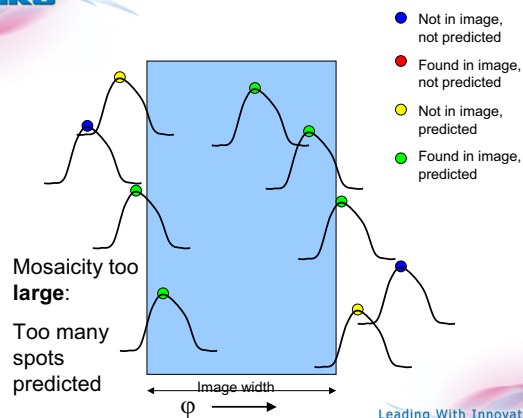
Leading With Innovation



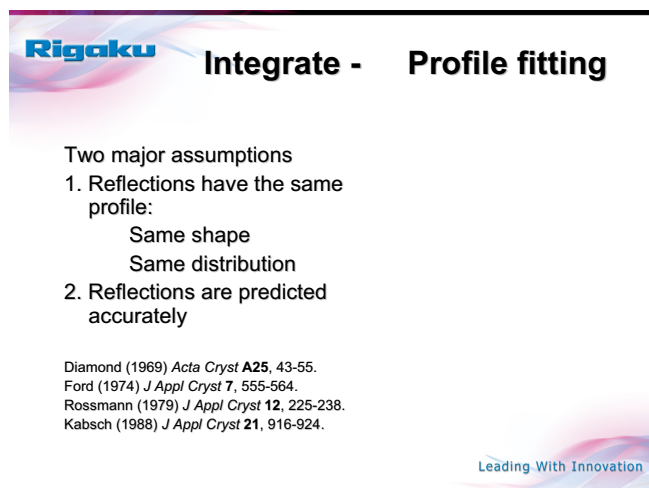
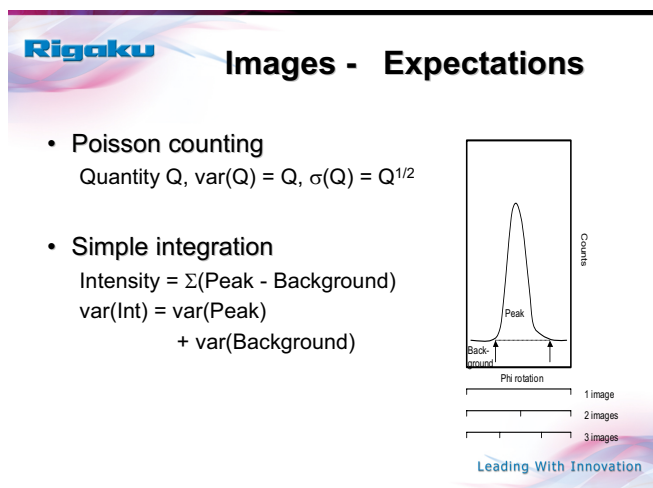
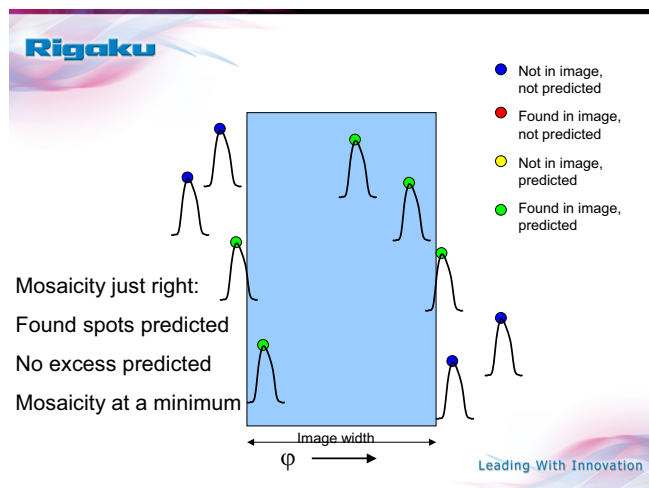
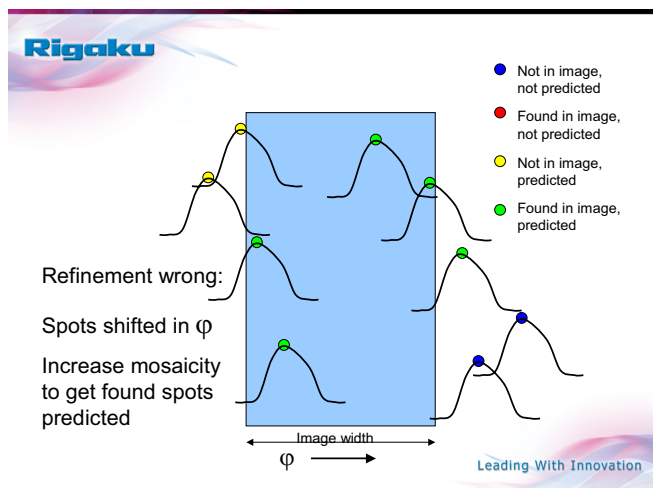
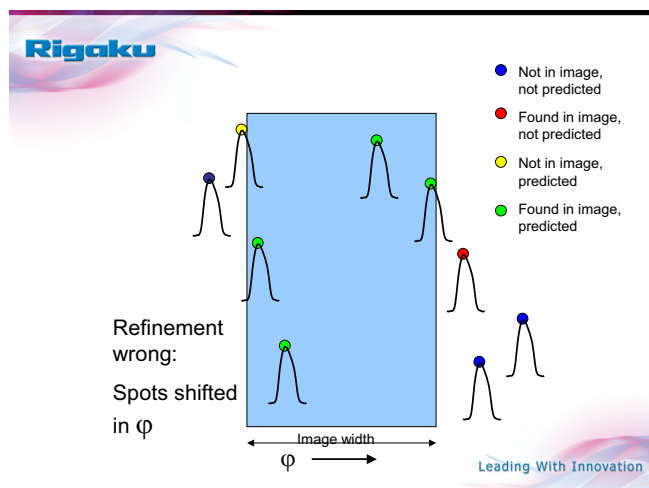
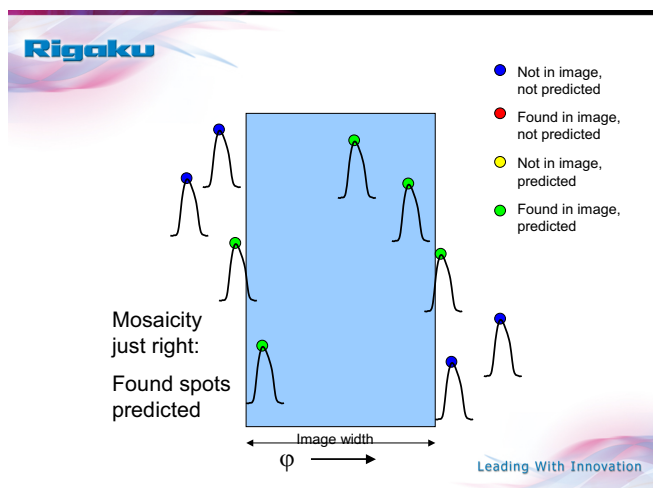
Leading With Innovation



Leading With Innovation



Leading With Innovation





Reference profile = Superposition of pixel values



Poor predictions  
= Poor superposition

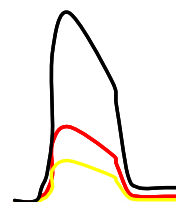


Good predictions  
= Good superposition



Two major assumptions

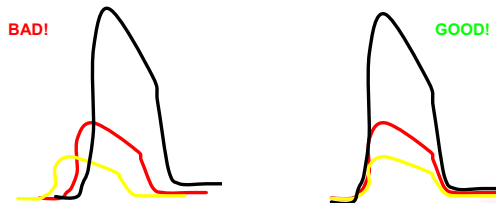
1. Reflections have the same profile:  
Same shape  
Same distribution
2. Reflections are predicted accurately





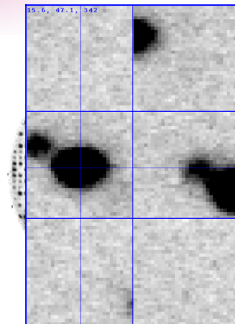
## Integrate - Profile fitting II

Bad predictions = Bad reference profile



Leading With Innovation

## Integrate - Profile fitting III



$$f = \sum_i \frac{(p_i - I_c)^2}{v_i}$$

$$I_{prof} = \frac{\sum_i \frac{p_i c_i}{v_i}}{\sum_i \frac{p_i^2}{v_i}}$$

Leading With Innovation

## Variations

Integrate in 2D, later post-refine and sum partials  
MOSFLM, denzo, HKL2000

Integrate in 3D, refine as you go along  
XDS, MADNES, d\*TREK

Box & spot size – user input or automatic; fixed or plastic

Leading With Innovation

## More details

How much are Bragg peaks rasterized?

What about powder rings such as from ice?

Wide slice 5 degree images?

Or fine slice 0.3 degree images?

Or 0.5 degree images?

What about systematic and erratic errors?

Bad pixels, shadows, moving shadows – mask them out

Zingers

$K\alpha_1/K\alpha_2$  at high  $2\theta$  - shift vectors are calculated and applied

Scale and get statistics as you go along

Update refinement continually

Detector gain

Spot overlap

Which bottle has naturale water and which has sparkling?

Leading With Innovation

## Scaling

- Correction of systematic errors
- Outlier rejection
- Validation of sigmas

$$\sigma_{adj}^2 = (\sigma_{in} E_{mul})^2 + (I_{in} E_{add})^2$$

Leading With Innovation

## Scaling

## Correction of systematic errors

- different crystal volumes
- different exposure times
- different detectors
- radiation damage
- wavelength dependent factors
- different or fluctuating source intensities
- different absorption due to different paths through the crystal and other matter

Leading With Innovation

## Atomic pair distribution function (PDF) analysis: What, When, Why, How?

**S.J.L. Billinge**

*Department of Physics and Astronomy,  
Michigan State University*

- PDF: what, when, why, how
- Software Projects and Software Engineering
  - DANSE project
- Some provocative remarks

<http://nirt.pa.msu.edu/>

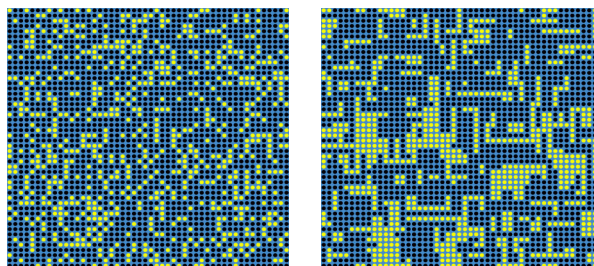
<http://nirt.pa.msu.edu/>

### PDF: why?

- When your crystals have defects that you care about
  - "Crystals are like people, it is their defects that make them interesting" (attributed to F. C. Franck)
  - Defects can be static, dynamic (phonons), chemical disorder, displacive disorder...
- When your crystals aren't crystals at all
  - Short-range order only (glasses, liquids)
  - Intermediate range order (nanoparticles, nanocrystallinity)
- Extracting this information is
  - Important
  - Difficult
  - Not crystallography (by definition, defects break periodicity), though MANY crystallographic concepts are fundamentally useful
  - "The nanostructure problem"

<http://nirt.pa.msu.edu/>

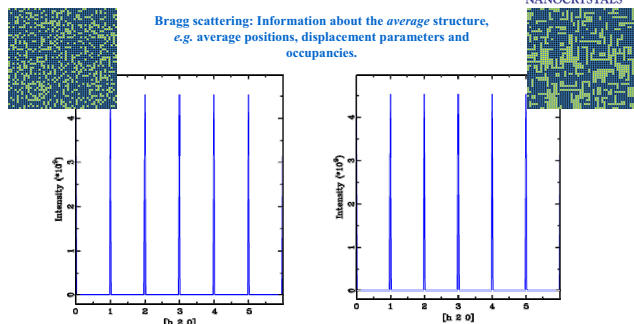
### PDF: what



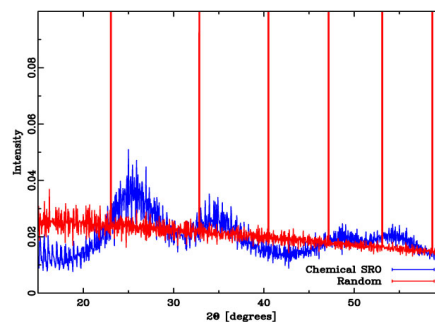
Cross section of 50x50x50 unit cell model crystal with 70% black atoms and 30% vacancies !  
Thanks to Thomas Proffen for the simulations!

<http://nirt.pa.msu.edu/>

### Bragg peaks are blind ..

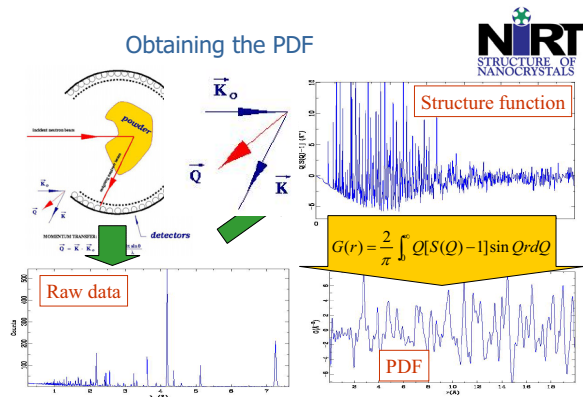


<http://nirt.pa.msu.edu/>



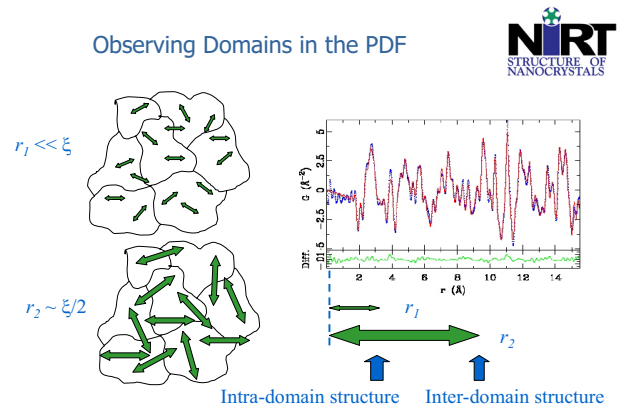
<http://nirt.pa.msu.edu/>

## Obtaining the PDF



<http://nirt.pa.msu.edu/>

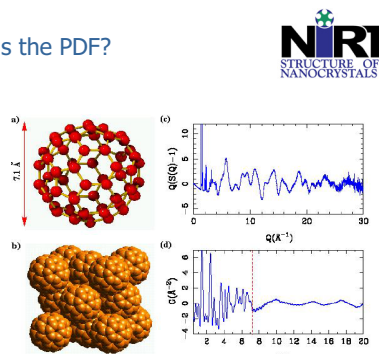
## Observing Domains in the PDF



<http://nirt.pa.msu.edu/>

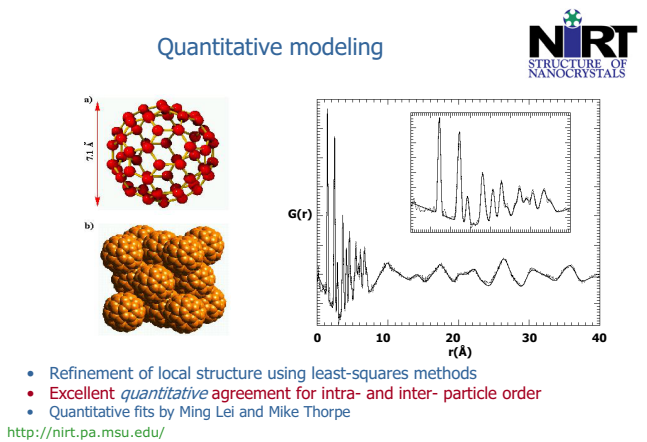
## What is the PDF?

- Sit on an atom and look at your neighborhood
- $G(r)$  gives the probability of finding a neighbor at a distance  $r$
- PDF is experimentally accessible



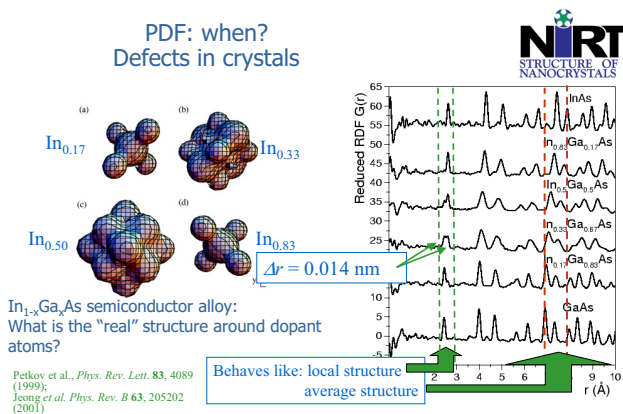
<http://nirt.pa.msu.edu/>

## Quantitative modeling



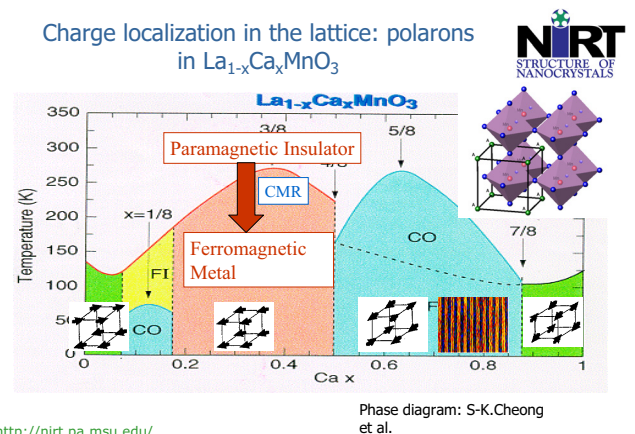
<http://nirt.pa.msu.edu/>

## PDF: when? Defects in crystals



<http://nirt.pa.msu.edu/>

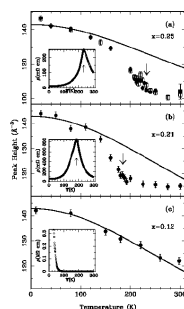
## Charge localization in the lattice: polarons in $\text{La}_{1-x}\text{Ca}_x\text{MnO}_3$



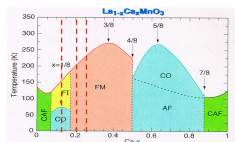
<http://nirt.pa.msu.edu/>



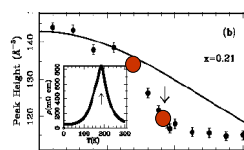
MI(T) transition is polaronic localization-delocalization transition



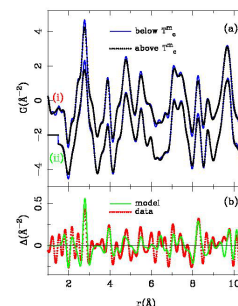
- PDF peaks are broad and low when polarons are present
- PDF peaks are narrow and sharp when polarons are absent
- SJLB *et al*, *Phys. Rev. Lett.* 77, 715 (1996).



What do the polarons look like?



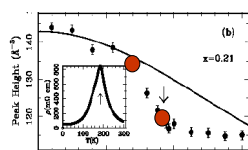
- How does the local structure change on going through the MI transition?



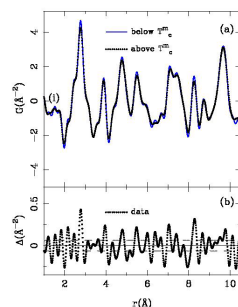
<http://nirt.pa.msu.edu/>

<http://nirt.pa.msu.edu/>

What do the polarons look like?



- How does the local structure change on going through the MI transition?
- Software for Data Transformations: corollary to Kevin Cowtan's talk yesterday

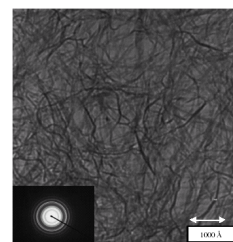
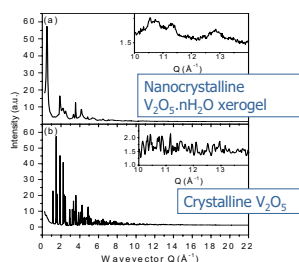


<http://nirt.pa.msu.edu/>

PDF when?  
When your chemistry colleagues give you muck like this:



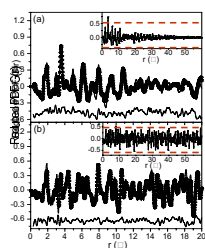
- Nanocrystalline materials:



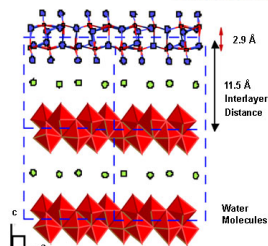
- Samples from the group of Mercuri Kanatzidis, MSU Chemistry

<http://nirt.pa.msu.edu/>

Structure of xerogel

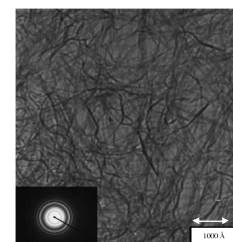
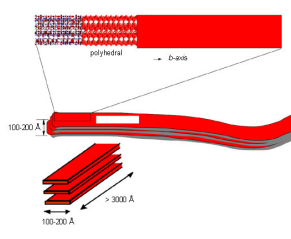


- Xerogel has bilayers of edge-shared VO<sub>5</sub> octahedra separated by water molecules
- Notice loss in peak amplitude above 11.5 Å => turbostratic disorder



<http://nirt.pa.msu.edu/>

"Nanostructure" in the xerogel

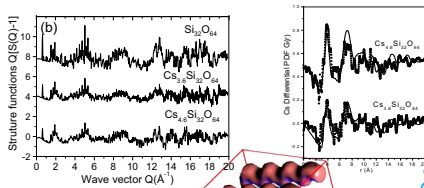


- Turbostratic disorder seen in the PDF consistent with bent and tangle fibres

V. Petkov, *et. al.*, *J. Am. Chem. Soc.* **121**, 10157 (2002).

<http://nirt.pa.msu.edu/>

## Structure of intercalants: inorganic electrode



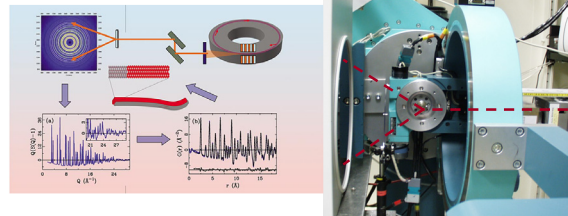
- Cs forms  $\text{Cs}^+$  in zig-zag pattern
- Electrons are counter-ions

- With Valeri Petkov, Tom Vogt and Dye group
- Zeolite ITQ-4 has 1D channels of  $\sim 7\text{\AA}$  diameter
- Cs is intercalated
- X-ray data from NSLS-X7A

Physical Review  
**Focus**  
Focus Archive PDB Index Image Index Focus Search  
Previous Story / Next Story / January-June 2002 Archive  
Phys. Rev. Lett. **89**, 075502  
(print issue of 12 August 2002)  
Title and Authors  
29 July 2002  
**Close-Up of Atomic Cave-Dwellers**

<http://nirt.pa.msu.edu/>

## RAPDF Geometry



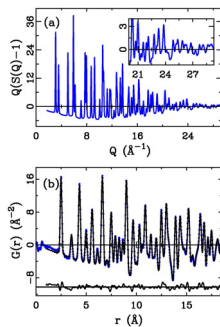
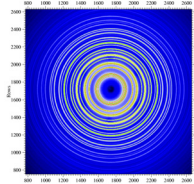
<http://nirt.pa.msu.edu/>

## Rapid Acquisition PDFs



### Fast x-ray PDFs

- Four orders of magnitude decrease in data collection time!
- Nickel data, 1s collection time,  $Q_{\text{max}} = 28\text{ \AA}^{-1}$
- Developed in collaboration with Xiangyun Qiu, Pete Chupas, Jon Hanson, Peter Lee and Clare Grey



<http://nirt.pa.msu.edu/>

## Computational issues: A Brief History of PDF



- Pieter Debye, 1912:

$$I = \sum_n \sum_m f_n f_m^* \frac{\sin qr_{nm}}{qr_{nm}}$$

- Fritz Zernike and Jon Prins, 1927:

$$4\pi r^2 \rho(r) = 4\pi r^2 \rho_a + \frac{2r}{\pi} \int_0^\infty q i(q) \sin qr dq$$



<http://nirt.pa.msu.edu/>

## History

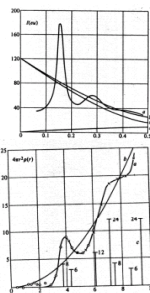


Debye and Menke, Z. Phys. (1930)

PDFs of mercury

Tarasov, L. P., and Warren, B. E., (1936) *J. Chem. Phys.*, **4**, 236.

X-ray PDFs of molten sodium



<http://nirt.pa.msu.edu/>

## History of PDF



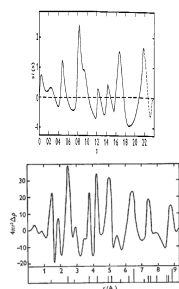
- Early 1930's
  - Computer: slide rule
  - Time to Fourier transform: few days
  - Time to paper: 6 months

<http://nirt.pa.msu.edu/>

## History



### Disordered Carbon



Warren, B. E., (1934) *J. Chem. Phys.* **2**, 551.

Franklin R. E. (1950) *Acta Crystallogr.* **3**, 107  
Franklin R. E. (1951) *Proc. R. Soc. London A.* **209**, 196



<http://nirt.pa.msu.edu/>

## History of PDF



- 1930's
  - Computer: slide rule
  - Time to Fourier transform: few days
  - Time to paper: 6 months
- 1950's
  - Computer: Beevers Lipson strips + pen + paper
  - Time to Fourier transform: "The whole procedure is very simple and it is readily performed in three or four hours"-B.E. Warren
  - Time to paper: 6 months

<http://nirt.pa.msu.edu/>

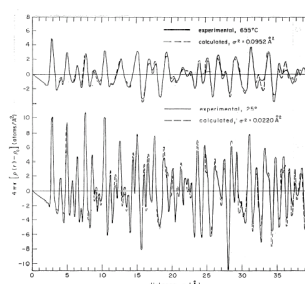
## Beevers Lipson strips



Gould BCA newsletter

<http://nirt.pa.msu.edu/>

## History



### PDFs from crystalline Aluminum

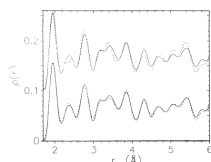
R. R. Fessler, Roy Kaplow and B. L. Averbach, *Phys. Rev.* **150**, 34 (1966).

### First use of Reverse-Monte-Carlo refinement

Kaplow R, Rowe, T. A. and Averbach, B. L. (1968), *Phys. Rev.* **168**, 1068.

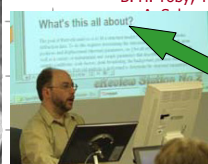
<http://nirt.pa.msu.edu/>

## 20 years later...



Combine Monte-Carlo modelling with crystalline PDFs to get real, quantitative, local structural information: the first such paper was on TI high-Tc superconductors

B. H. Toby, T. Egami, J. D. Jorgensen, and M. M. Perlman, *Phys. Rev. Lett.* **64**, 2414-10 (1990)



And yet 10 years on Brian's still confused about the subject

<http://nirt.pa.msu.edu/>

## History of PDF



- 1930's
  - Computer: slide rule
  - Time to Fourier transform: few days
  - Time to paper: 6 months
- 1950's
  - Computer: Beevers Lipson strips + pen + paper
  - Time to Fourier transform: "The whole procedure is very simple and it is readily performed in three or four hours"-B.E. Warren
  - Time to paper: 6 months
- 1980's
  - Computer: DEC microvax
  - Time to Fourier transform: ~15 mins.
  - Time to paper: 6 months

<http://nirt.pa.msu.edu/>

## PDF of crystals: the early days



- Microvax: 16Mb memory, 100Mb hard drive
- PDFvax: 6 students, 2 postdocs, no crashes
- Picture credit: Tom Carlson  
Location: Williamsburg, VA
- Well, my employer was re-modeling the basement and they were going to **throw it out!** Look at it! Would you let them just toss it! I think not. (And to think, they kept the AS/400! What were they thinking?) So I somehow wedged both towers into my Volkswagen and went to pick up my wife at her work. I could tell you what she said, but I like schools to be able to link to here.

<http://nirt.pa.msu.edu/>

## History of PDF



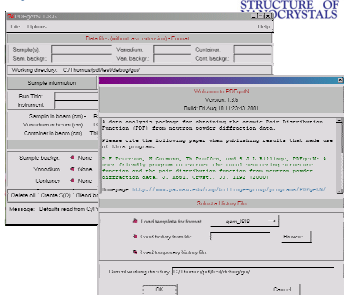
- 1930's
  - Computer: slide rule
  - Time to Fourier transform: few days
  - Time to paper: 6 months
- 1950's
  - Computer: Beavers Lipson strips + pen + paper
  - Time to Fourier transform: "The whole procedure is very simple and it is readily performed in three or four hours"-B.E. Warren
  - Time to paper: 6 months
- 1980's
  - Computer: DEC microvax
  - Time to Fourier transform: ~15 mins.
  - Time to paper: 6 months
- 2000's
  - Computer: 1.5GHz Pentium PC
  - Time to Fourier transform: <1 second
  - Time to paper: 6 months

<http://nirt.pa.msu.edu/>

## PDF how: Data Analysis software



- E.g., PDFgetN, PDFgetX2. Available from ccp14 and: <http://www.ccp14.ac.uk/ccp14.cgi>
- <http://nirt.pa.msu.edu/>
- Graphical user interface & integrated plotting.
- Supports most TOF neutron powder file formats.
- Records all processing parameters as part of output files G(r) and S(Q).
- Runs on Windows 95/98/NT/2000 and UNIX
- Legacy code, wrapped by Pete Peterson and Thomas Proffen



Peterson et al., *J. Appl. Cryst.* **33**, 1192 (2000)

<http://nirt.pa.msu.edu/>

## Modelling software, e.g., PDFFIT – real space Rietveld ..



- Program features
  - Controlled by FORTRAN style command language including loops and IF statements.
  - User defined relation between parameters and refinement variables.
  - Multiple structural phases supported.
  - Multiple data sets (neutron and X-ray) supported.
  - Interfaces with DISCUS, KUPLOT and ATOMS.
  - Available from ccp14 and: <http://www.totalscattering.org>
  - Online help function.



Proffen and Billinge, *J. Appl. Cryst.* **32**, 572 (1999)

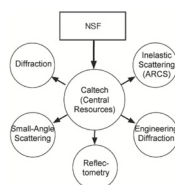
<http://nirt.pa.msu.edu/>

- Technical information
  - UNIX or Windows operating system.
  - Binary or source code distribution.
  - Written in FORTRAN-77 (and some C).
  - Written by Thomas Proffen after an earlier effort by Simon Billinge

## What is DANSE?



- It stands for Distributed Data analysis for neutron scattering experiments
- It is a software proposal for \$13M to the NSF
- It has received one year of design funding (~\$1M)
- There will be a funding decision for construction funding in November
- Lead PI's Michael Aivazis and Brent Fultz (Caltech)

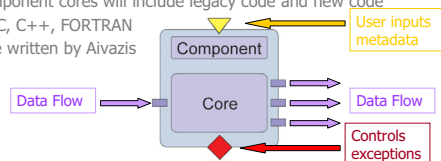


<http://nirt.pa.msu.edu/>

## DANSE basic philosophy

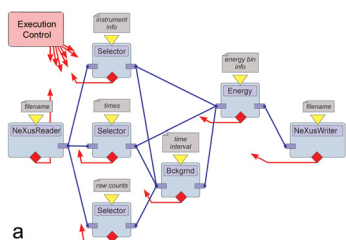


- Software Component architecture design
  - Maintainable, extensible, reusable code
- Framework is Pyre, python based wrappers that will support distributed computing and data-streaming
  - Combines speed of compiled languages (component cores) with flexibility of scripting languages (python)
  - Component cores will include legacy code and new code
    - C, C++, FORTRAN
  - Pyre written by Aivazis



<http://nirt.pa.msu.edu/>

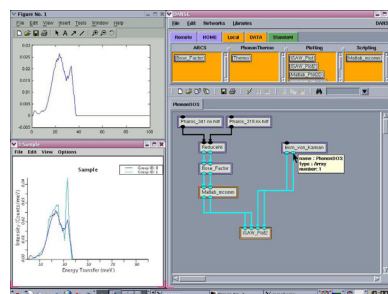
## Applications are networks of components



- Schematic of an application to produce an energy spectrum from inelastic scattering tof neutron data
- Components themselves can be nested
- Framework handles things such as exceptions and validation of inputs

<http://nirt.pa.msu.edu/>

## Cobra/Viper GUI



- GUI is obsolete, but gives an idea of how this can work
- Directories reside on different computers, components are dragged and dropped onto the desktop and wired together
- Users interact with the software on different levels: novice, senior scientist, component developer, framework maintainer/developer
- Cobra/Viper and independent development

<http://nirt.pa.msu.edu/>

## Provocative remark 1: Inheritance, code design and UML



- Scientific Programs are algocentric
  - Start with the algorithm, build out
- Commercial software is usercentric
  - Need to sell the software
  - Need to figure out what the people will buy
  - Need to build that
- Determining "Use Cases" is the business of finding out what people want and how they will use the code
- Once you have that, you build the code to deliver what is wanted
- UML diagrams can help

<http://nirt.pa.msu.edu/>

## Provocative remark 2: Inheritance, code design and UML



```

If __name__ == '__main__':
    ...
    greatData1=science.getData(student1)
    greatResults1=science.analyzeData(student1,
    greatData1)
    Nature=science.writePaper(student1,greatRes
    ults1)
    ...
    Science=science.writePaper(student1,greatRe
    sults3)
    ...
    greatData4=science.getData(student2)
    greatResults4=science.analyzeData(student2,
    greatData4)
    PRL=science.writePaper(student2,
    greatResults4)

    simonRich=rewards.payraise(simon)
    simonFamous=prize.nobel(simon)
    
```

<http://nirt.pa.msu.edu/>

## Inheritance, code design and UML

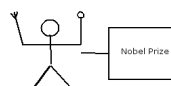


```

Class Science()
    def __init__()
    def getData()
    def analyzeData()
    def writePaper()
    def giveTalk()
Class rewards(science)
    def __init__()
    def payRaise()
Class Prize(rewards)
    def __init__()
    def thesis()
    def nobel()
    def knighthood()
    
```

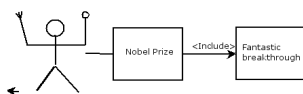
<http://nirt.pa.msu.edu/>

## What can UML do for us?



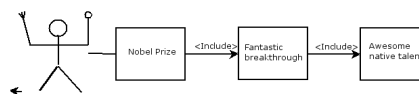
<http://nirt.pa.msu.edu/>

What can UML do for us?



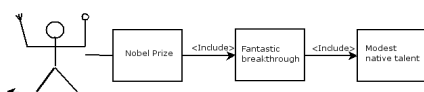
<http://nirt.pa.msu.edu/>

What can UML do for us?



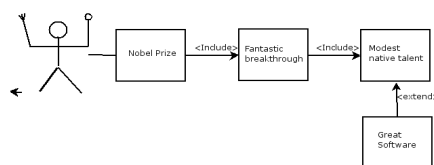
<http://nirt.pa.msu.edu/>

What can UML do for us?



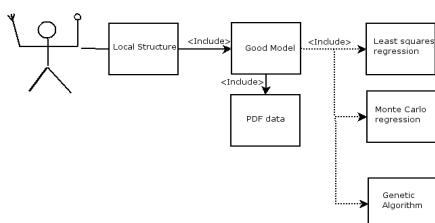
<http://nirt.pa.msu.edu/>

What can UML do for us?



<http://nirt.pa.msu.edu/>

What can UML do for us?



<http://nirt.pa.msu.edu/>

Summary



- PDF: what, when, why, how
  - Disorder in crystals
  - Nanocrystals, nanoparticles and nanostructured materials
- Software Projects and Software Engineering
  - Component architectures
  - DANSE project
- Some provocative remarks

Useful introductory paper:

S. J. L. Billinge and M. G. Kanatzidis **Beyond crystallography: the study of disorder nanocrystallinity and crystallographically challenged materials**, *Chem. commun.*, 749-760 (2004).

<http://nirt.pa.msu.edu/>

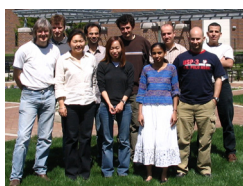
## Useful book



<http://nirt.pa.msu.edu/>

<http://nirt.pa.msu.edu/>

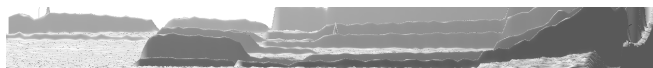
## Acknowledgements



- Thomas Proffen (former post-doc, now LANL)
- Pete Peterson (former student, now ORNL)
- Pavol Juhas (current post-doc)
- Valeri Petkov (former post-doc, now at CMU)
- Xiangyun Qiu (former student, now at Cornell)
- Mike Thorpe (ASU), Valentin Levashov, Ming Lei
- Groups of Mercouri Kanatzidis, Jim Dye and Tom Pinnavaia
- Tom Vogt
- Pete Chupas, Jon Hanson, Peter Lee and Clare Grey
- **Facilities:**
  - APS, CHESS, NSLS (and people therein)
  - MLNSC, ISIS, IPNS (and people therein)
- **Funding:** NSF-DMR 0304349, NSF-DMR 0075149, CHE-0211029, DOE-DE-FG02-97ER45651

<http://nirt.pa.msu.edu/>

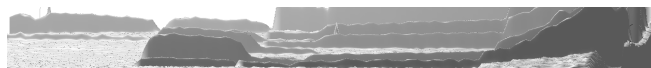




## Dealing with overlapped data

Bill David, ISIS Facility,  
Rutherford Appleton Laboratory, UK

IUCr Computing School Siena 18-23 August 2005



## Powder diffraction: issues and algorithms

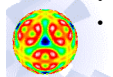
Bill David, ISIS Facility,  
Rutherford Appleton Laboratory, UK

IUCr Computing School Siena 18-23 August 2005



## WIFD - standard disclosure

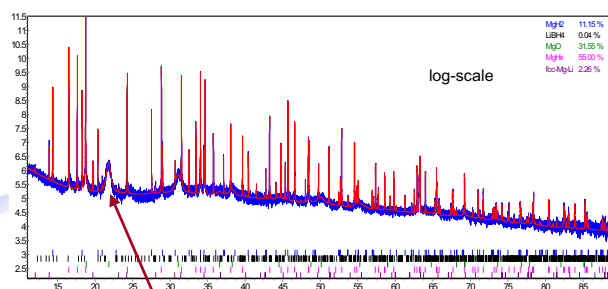
- 1983-5 (Oxford to ISIS)
  - GENIE – data manipulation and analysis package
    - based on VMS command line interpreter – still in use
    - (I still have my VAX (called JARAK) in the basement )
- 1983-
  - CCSL – FORTRAN77 crystallographic subroutine library
  - ([www.iill.fr/dif/ccsl/html/ccsldoc.html](http://www.iill.fr/dif/ccsl/html/ccsldoc.html)) Rubbia effect
  - basis of all Rietveld analysis at RAL until 1992
- 1997-
  - DASH - structure solution from powders
    - SA5TOR (VAX VMS) -2 weeks
    - GUI (Winteracter – all FORTRAN – 6 months)
    - CCDC –  $\alpha$  and  $\beta$  testing – 18 months



IUCr Computing School Siena 18-23 August 2005

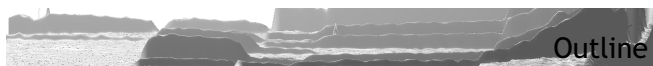


## Major advances in instrumentation

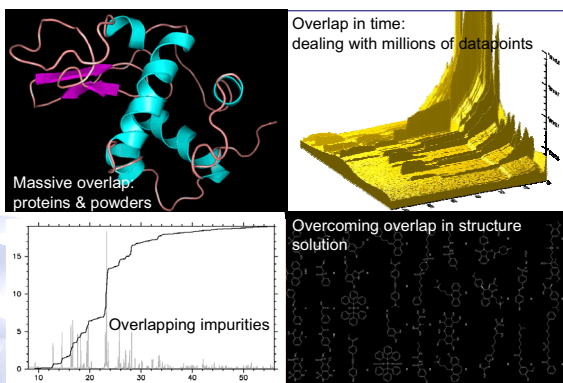


Note width differences

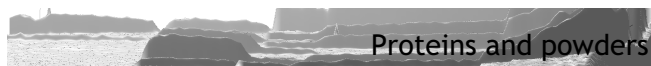
IUCr Computing School Siena 18-23 August 2005



## Outline



IUCr Computing School Siena 18-23 August 2005

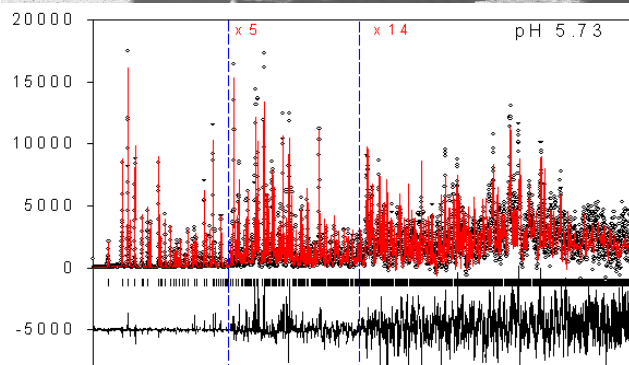


## Proteins and powders





## Proteins and powders

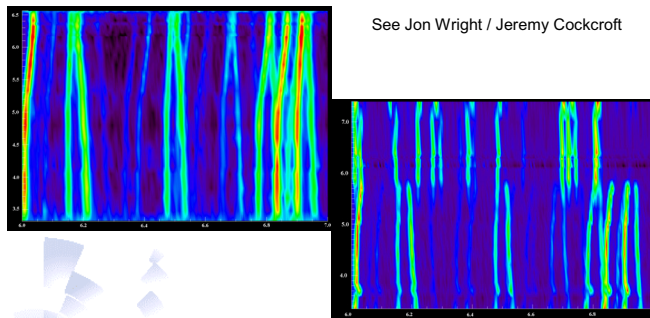


IUCr Computing School Siena 18-23 August 2005

ISIS

## Proteins and powders

See Jon Wright / Jeremy Cockcroft

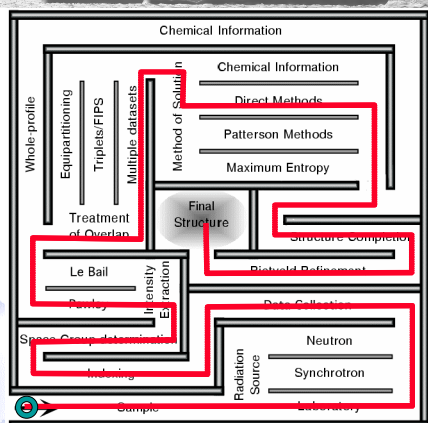


Colour representation of ID31 powder diffraction data from the pH variation experiment, from pH 6.56 – 3.33 of HEWL crystallised at (a) 4°C and (b) RT. At low temperature the tetragonal phase is favoured and a smooth anisotropic shift in the peak position is apparent.

IUCr Computing School Siena 18-23 August 2005

ISIS

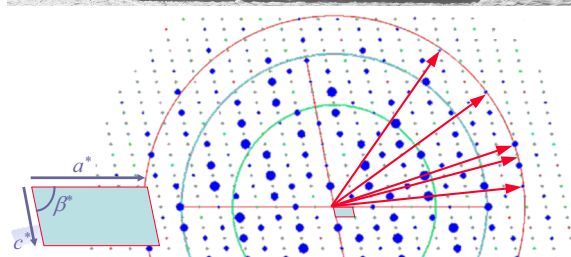
## powder diffraction - standard disclosure



after  
Baerlocher &  
McCusker

ISIS

## bottlenecks in the maze



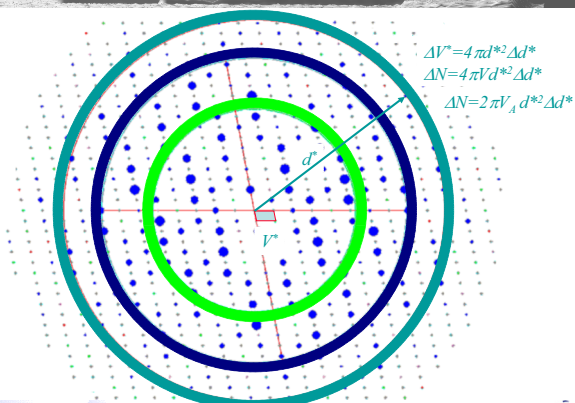
powder diffraction  
radial reciprocal space distance only -  $d$ -spacing

$$Q_{hkl} = h^2 a^{*2} + k^2 b^{*2} + l^2 c^{*2} + 2klb^*c^*\cos\alpha^* + 2hla^*c^*\cos\beta^* + 2hka^*b^*\cos\gamma^*$$

IUCr Computing School Siena 18-23 August 2005

ISIS

## TRAFFIC JAM in the maze!



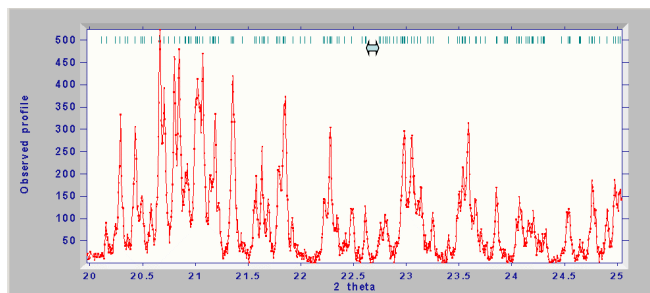
IUCr Computing School Siena 18-23 August 2005

ISIS

## peak density - TRAFFIC JAM in the maze!

$$x = \Delta 2\theta / \langle \Delta 2\theta \rangle = \Delta N(2\theta) \times \Delta 2\theta$$

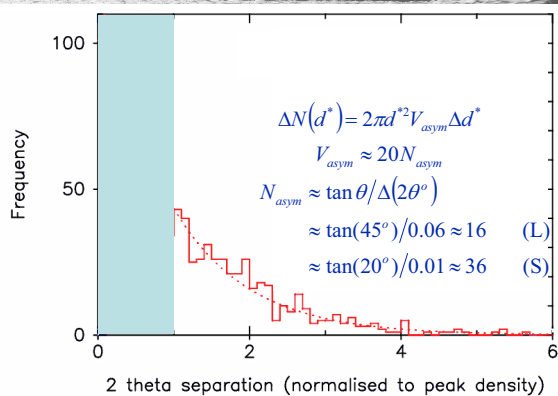
$$p_{NN}(x) \propto \exp(-x)$$



IUCr Computing School Siena 18-23 August 2005

ISIS

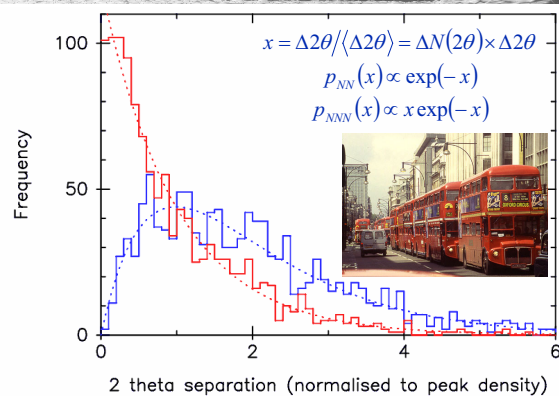
## peak density - TRAFFIC JAM in the maze!



IUCr Computing School Siena 18-23 August 2005

ISIS

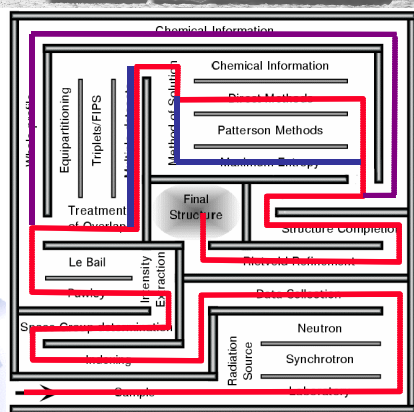
## peak density - TRAFFIC JAM in the maze!



IUCr Computing School Siena 18-23 August 2005

ISIS

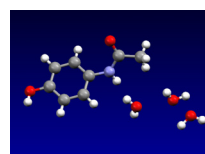
## dealing with the TRAFFIC JAM of peak overlap



IUCr Computing School Siena 18-23 August 2005

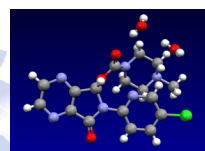
ISIS

## Dehydration of pharmaceutical compounds



### Paracetamol hydrates

$C_8H_9NO_2 \cdot nH_2O$   
pain-killer, analgesic, antipyretic  
4'-hydroxyacetanilide,  
acetaminophen, tylenol



### Zopiclone hydrates

$C_{17}H_{17}ClN_5O_3 \cdot 2H_2O$   
hypnotic - insomnia  
line phases: dihydrate - anhydrous

IUCr Computing School Siena 18-23 August 2005

ISIS

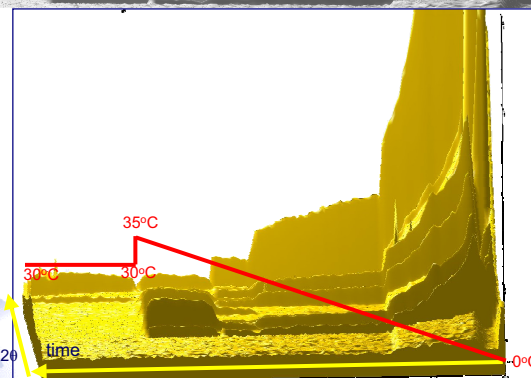
## Key points

- Analysing all the data as fully as possible
  - Managing a million data-points
    - 130 patterns
    - 8520 points per pattern
    - 1,107,600 points
- Identifying change
  - Principal component analysis / clustering
- Quantitative phase analysis
- Structure determination
- Rietveld refinement
  - Structure, microstructure & inhomogeneity

IUCr Computing School Siena 18-23 August 2005

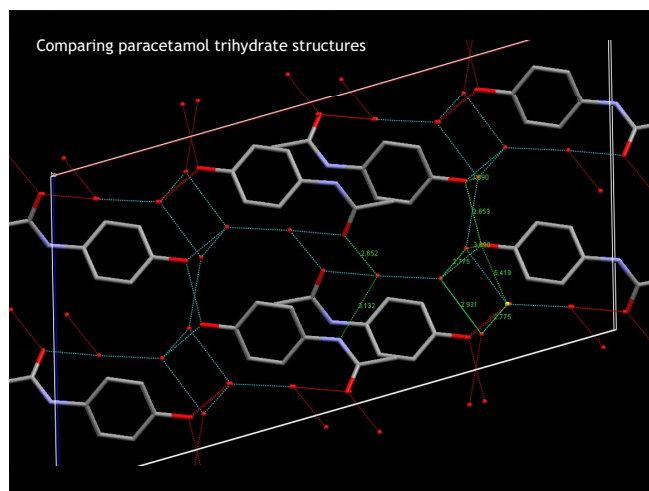
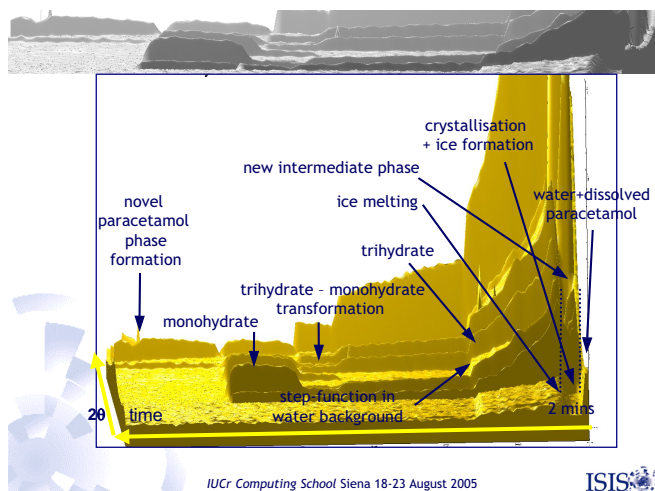
ISIS

## Key points



IUCr Computing School Siena 18-23 August 2005

ISIS



### Dehydration of pharmaceutical compounds

**Paracetamol hydrates**

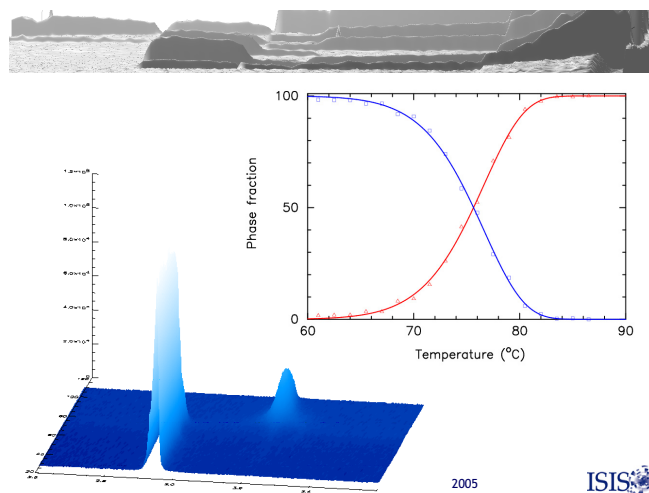
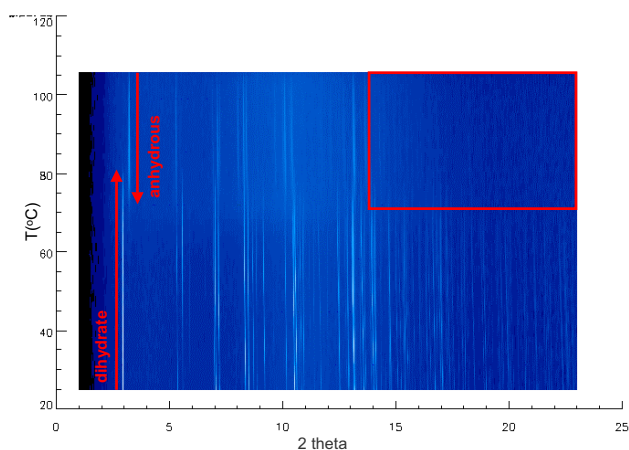
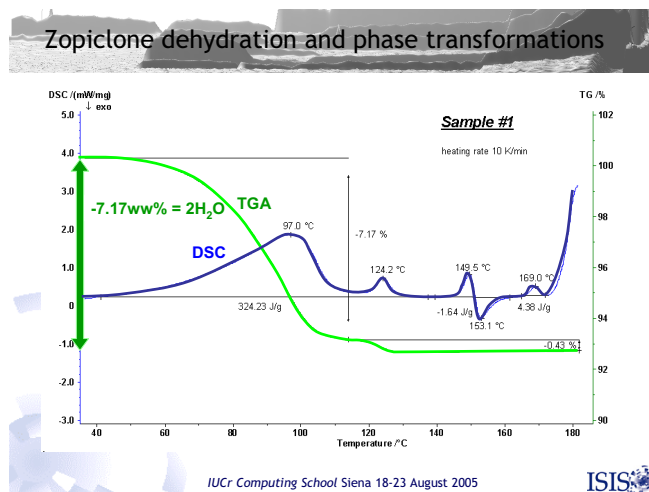
$C_8H_9NO_2 \cdot nH_2O$   
pain-killer, analgesic, antipyretic  
4'-hydroxyacetanilide,  
acetaminophen, tylenol

**Zopiclone hydrates**

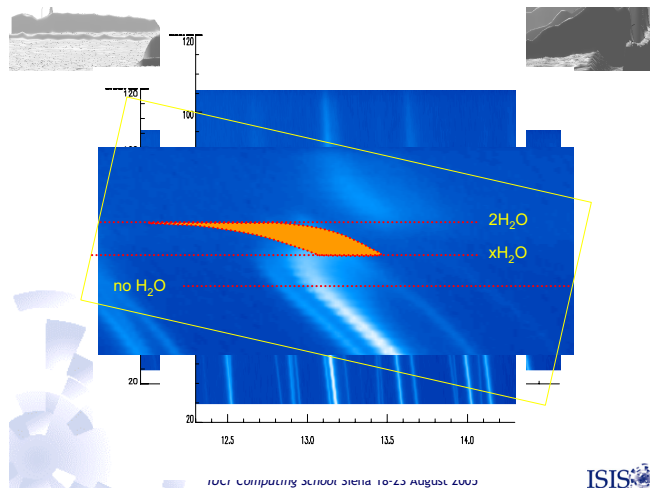
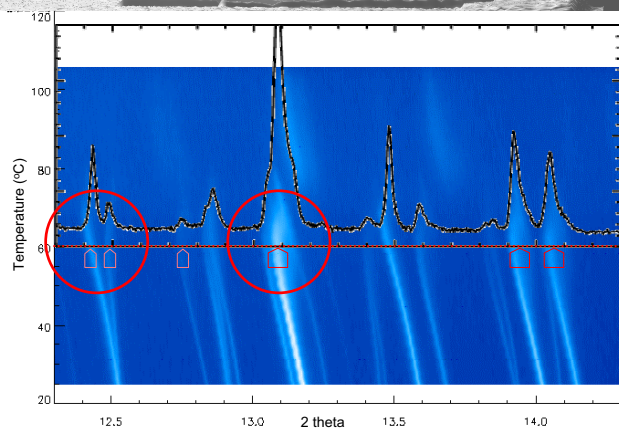
$C_{17}H_{17}ClN_5O_3 \cdot 2H_2O$   
hypnotic - insomnia  
line phases: dihydrate - anhydrous

IUCr Computing School Siena 18-23 August 2005

ISIS

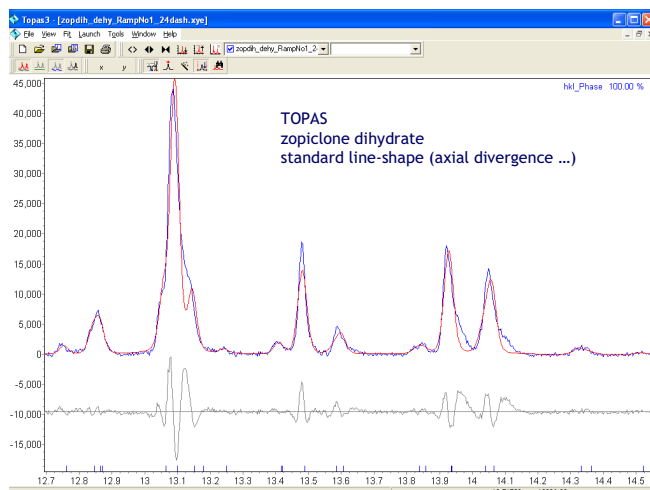
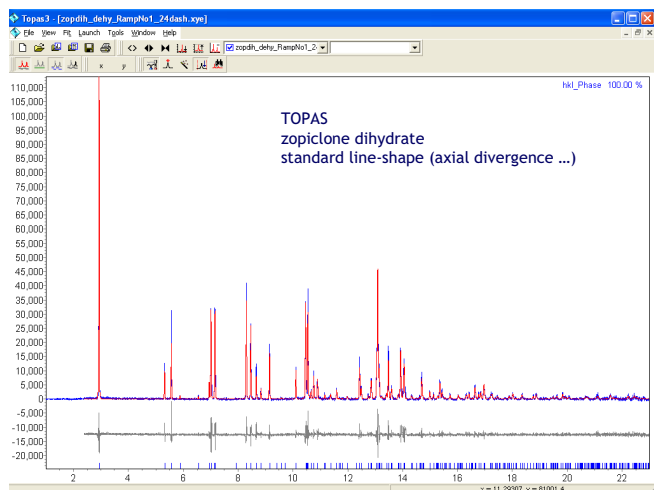


## Complex anisotropic sample line-shape

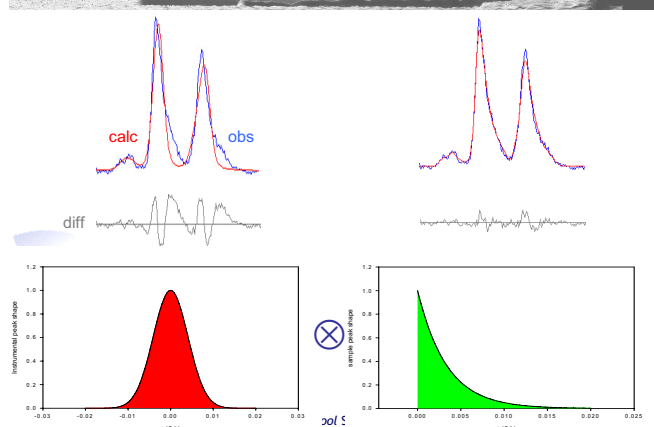


IUCr Computing School Siena 18-23 August 2003

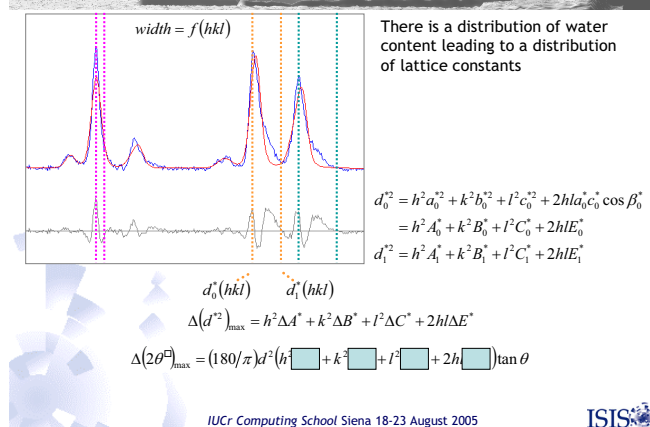
ISIS



## Complex anisotropic sample line-shape



## Complex anisotropic sample line-shape



ISIS



## Complex anisotropic sample line-shape

$$\Delta(2\theta)_{\max} = (180/\pi)d^2(h^2 + k^2 + l^2) + 2h \tan \theta$$

we have defined the limits of the sample line-shape

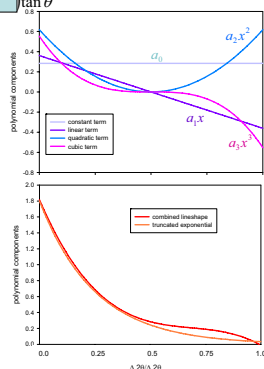
but we don't know the lineshape

construct a generalised lineshape using polynomials / orthogonal polynomials

cubic polynomial

$$\Delta(2\theta) = a_0 + a_1x + a_2x^2 + a_3x^3$$

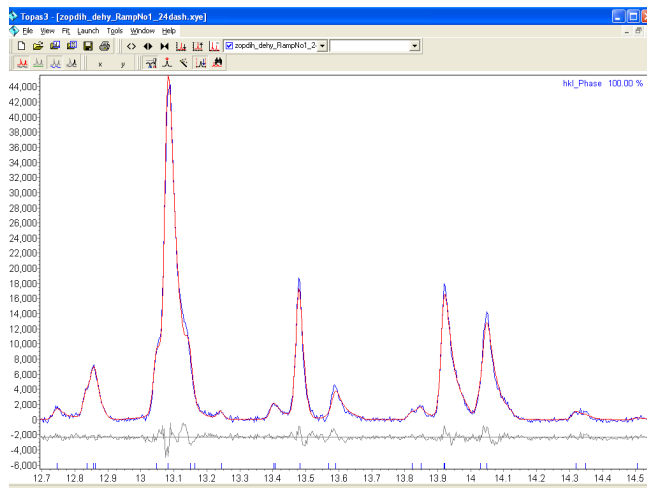
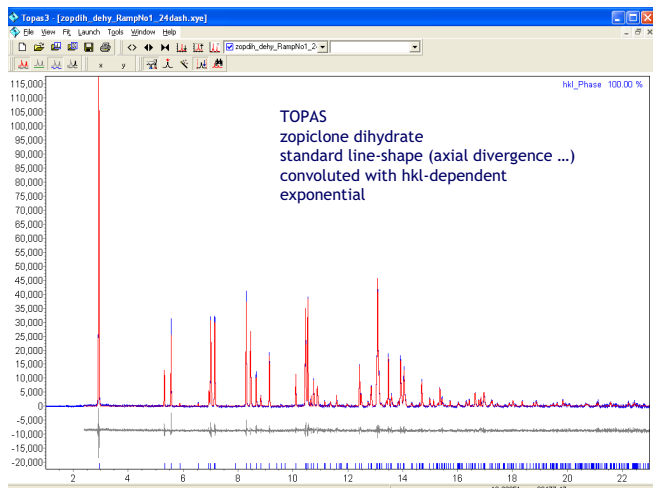
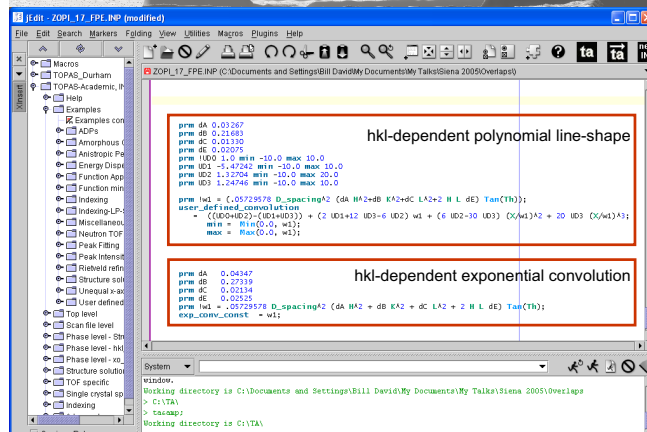
$$(2x-1) = \Delta(2\theta)/\Delta(2\theta)_{\max}$$



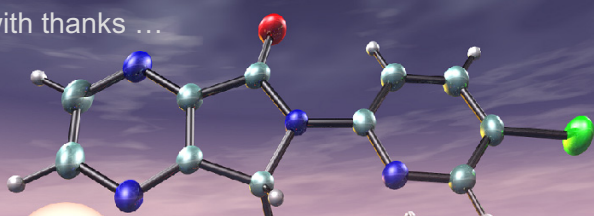
IUCr Computing School Siena 18-23 August 2005

ISIS

## TOPAS screenshot



with thanks ...



M Brunelli, A Fitch, J Wright (ESRF)  
A Coelho  
N Shankland, A Kennedy (Strathclyde)  
C Pulham (Edinburgh)  
K Shankland, A J Markvardsen (ISIS)

Zopiclone

## Programming the Science of Crystallography

PLATON, a Multipurpose  
Crystallographic Tool  
Ton Spek, Utrecht University

## Programming Languages

- Current choices are Fortran-(xx), C(++) or one of the many scripting languages (e.g. Python).
- My choice for scientific software over the last 30 years was and still is **Fortran**.  
I have seen many (scripting) languages come and go ...  
algol, pascal, ratfor ... and changed only once ...
- I might consider a conversion to C++ after my official retirement in 2009 (assuming that C++ is still mainstream by that time and not superseded by Fortran2xxx .....

## Pro's and Con's of Fortran

- **Fortran Pro's:**
  - Designed for scientific computing, readily available and still evolving to include additional useful constructs.
  - Relatively easy to learn and port to other platforms.
- **Fortran Con's:**
  - No longer mainstream in the current software development community.
  - Interface to C libraries (e.g. Xlib) needed for graphics functionality.

## PLATON AS AN EXAMPLE

- PLATON is focused mainly on small-molecule applications.
- The development of PLATON is essentially evolutionary, science driven and based on the needs of a national single crystal structure facility.
- Following is an overview of the IDEAS and TOOLS that have been implemented over the past 25 years in the program suite PLATON.

## PLATON IMPLEMENTATION

- The development of PLATON started on various CDC mainframe platforms and migrated via VAX/VMS and DEC-UNIX to the PC/LINUX platform.
- Implementations are also available for MS-WINDOWS (thanks to Louis Farrugia, Glasgow) and Mac-OSX.
- PLATON tries to be compatible and complementing to the SHELX software suite.
- PLATON is currently used as the major structure validation engine in the IUCr CHECKCIF facility.

## PLATON ORGANISATION

- Single FORTRAN source + a small C routine as an interface to X11 graphics.
- Separate group of routines for the handling of the Space Group Symmetry.
- Separate group of routines for handling the Graphics (X11/PostScript/HPGL).
- Separate group of reusable global routines (SORT, INVERT, etc.)

## Input Files

### Input files are:

1. A parameter/coordinate file of type *res*, *cif*, *fdat*, *spf*. The file type is guessed from the content, not from the file extension.
2. A reflection file of type *hkl* or *fcf*.
3. Command line input for instructions.

## Output Files

- A full listing file (.lis).
- The PostScript version of .lis (.lps) for printing on a laserprinter or viewing with GhostScript.
- A summary listing on the console.
- Optionally a new parameter file
- Optionally a new reflection file
- Optionally a validation report (.chk, .fck)

## Graphics Output

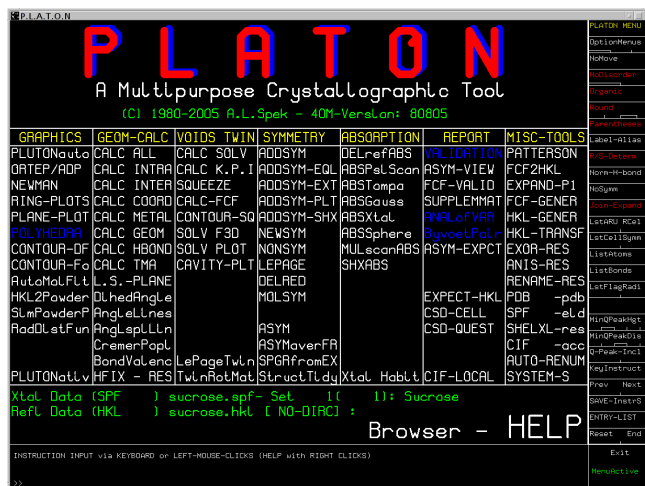
- Graphics output is implemented via calls to a single routine.
- This routine implements graphics instructions for the various types of graphics hardware.
- Currently, only X11, PostScript and HPGL are supported.
- In the past there was similar support for Tektronix etc.
- X11 library calls are implemented in a single C routine.
- The Windows version substitutes its own library calls.
- PLATON implements its own character set.

## Features

- PLATON includes a number of unique tools such as ADDSYM, VOIDS, SQUEEZE, TwinRotMat, CIF, FCF Validation, BijvoetPairs, and SYSTEM-S.
- Provides a 'research framework' for the convenient implementation and testing of new ideas.
- Few outside dependencies (single source) (libX11 or equivalent for graphics).
- Non-standard language features are avoided.
- Up-to-Date HTML-HELP (via right mouse click on item) with a browser over the Internet or locally installable.

## Entry points

- Via command line options allowing for use in scripts:  
e.g. '**platon -u shelxl.cif**' will produce as the only output a file '**shelxl.chk**' with a validation report.
- The clickable PLATON main menu gives an overview of the available functions.



## Space Group Symmetry

- 230 Unique Space Groups, multiple settings, synonyms, specification.
- Explicit symmetry operator, H-M or Hall Symbol input
- Space Group Routine: Multiple callable functions:
  - Expansion of the set of symmetry generators
  - Explicit symmetry → H-M and Hall Symbol
  - Symmetry operations on coordinates or reflection h,k,l
  - Multiplication of two supplied symmetry operators  
 $(R'|t') = (R1|t1)(R2|t2) \rightarrow \text{Network Analysis}$
  - Return inverted symmetry operation (including transl.)

## Geometry Analysis

- **Intra-molecular geometry**  
bonds, angles, torsions, rings, planes etc.
- **Inter-molecular geometry**  
Short contacts, H-bonds, networks
- **Coordination geometry**

Default: CALC ALL



## Derived Geometry and Standard Uncertainties

- Standard uncertainties for derived quantities  $f(p)$  can be derived in principle using the Least-Squares Covariance Matrix and the expression for the propagation of error:

$$\sigma^2(f) = \sum_{ij} (\delta f / \delta p(i)) (\delta f / \delta p(j)) \text{cov}(p(i), p(j))$$

- Or in case only variances are available:

$$\sigma^2(f) = \sum_i (\delta f / \delta p(i))^2 \sigma^2(p(i))$$

- **Analytical Evaluation** (clumsy for torsion angles and up)
- **Numerical**: approximate  $\delta f / \delta p(i) \sim (f(p + \Delta i) - f(p)) / \Delta i$

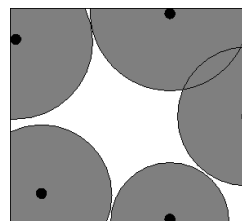
Take:  $\Delta i = \sigma(p(i))$ , then

$$\sigma^2(f) \sim \sum_i ((f(p + \sigma(p(i))) - f(p))^2$$

## Solvent Accessible Voids

- A typical crystal structure has only 65% of the available space filled.
- The remainder volume is in voids (cusps) in-between atoms (too small to accommodate an H-atom)
- **Solvent accessible voids** can be defined as regions in the structure that can accommodate at least a sphere with radius 1.2 Angstrom without intersecting with any of the van der Waals spheres assigned to each atom in the structure.
- Algorithm: Graphical and Computational

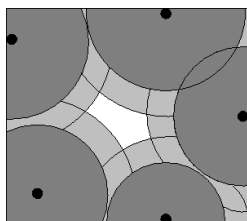
## DEFINE SOLVENT ACCESSIBLE VOID



STEP #1 – EXCLUDE VOLUME INSIDE THE VAN DER WAALS SPHERE

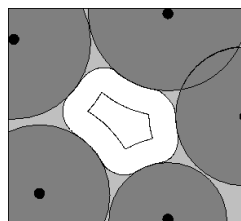


DEFINE SOLVENT ACCESSIBLE VOID



STEP # 2 – EXCLUDE AN ACCESS RADIAL VOLUME  
TO FIND THE LOCATION OF ATOMS WITH THEIR  
CENTRE AT LEAST 1.2 ANGSTROM AWAY

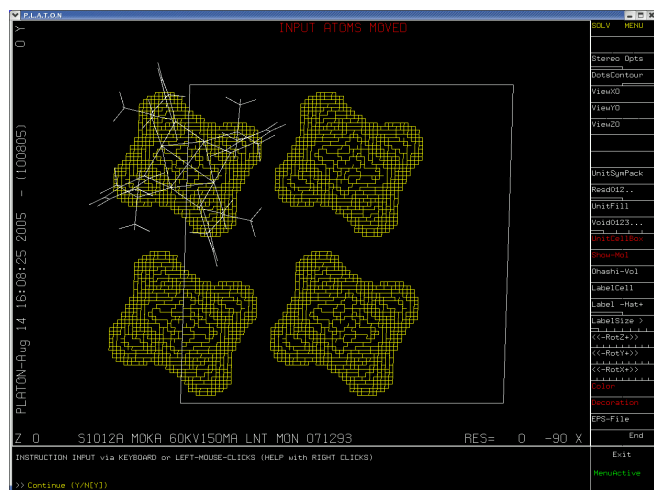
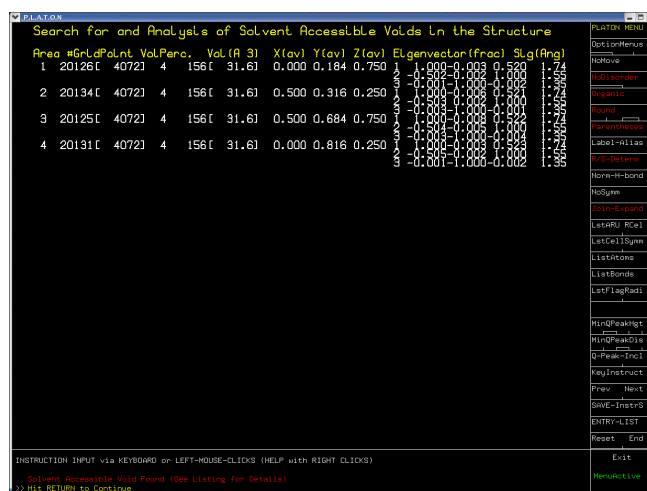
DEFINE SOLVENT ACCESSIBLE VOID



### STEP # 3 – EXTEND INNER VOLUME WITH POINTS WITHIN 1.2 ANGSTROM FROM ITS OUTER BOUNDS

## Voids: Algorithm

1. Expand the unitcell contents to P1
2. Define a 3D grid with gridstep  $\sim 0.2$  Angstrom and with the number of gridpoints in each direction a multiple of 12 (for exact symmetry mapping)
3. Scan through all gridpoints in search of gridpoints that have a distance greater than the probe radius to the nearest van der Waals sphere.
4. Join gridpoints into connected sets (S).
5. Expand this set with gridpoints within the probe radius from the surface of S.



## VOID APPLICATIONS

- Calculation of Kitaigorodskii Packing Index
- As part of the SQUEEZE routine to handle the contribution of disordered solvents in crystal structure refinement
- Determination of the available space in solid state reactions (Ohashi)
- Determination of pore volumes, pore shapes and migration paths in microporous crystals

## SQUEEZE

- Takes the contribution of disordered solvents to the calculated structure factors into account by back-Fourier transformation of density found in the 'solvent accessible volume' outside the ordered part of the structure.
- Filter: Input shelxl.res & shelxl.hkl  
Output: 'solvent free' shelxl.hkl
- Refine with SHELXL or Crystals

## Comment

- The Void-map can also be used to count the number of electrons in the masked volume.
- A complete dataset is required for this feature.
- Ideally, the solvent contribution is taken into account as a fixed contribution in the Structure Factor calculation (CRYSTALS) otherwise it is subtracted temporarily from  $F(\text{obs})^2$  (SHELXL) and reinstated afterwards for the final  $F_o/F_c$  list.

## Example

- Structure refined to  $R = 20\%$  in P-3
- Run TwinRotMat on CIF/FCF
- Result: Twinlaw with estimate of the twinning fraction and drop in R-value

## SQUEEZE Algorithm

1. Calculate difference map (FFT)
2. Use the VOID-map as a mask on the FFT-map to set all density outside the VOID's to zero.
3. FFT-1 this masked Difference map -> contribution of the disordered solvent to the structure factors
4. Calculate an improved difference map with  $F(\text{obs})$  phases based on  $F(\text{calc})$  including the recovered solvent contribution and  $F(\text{calc})$  without the solvent contribution.
5. Recycle to 2 until convergence.

## (Pseudo)Merohedral Twinning

- Options to handle twinning in L.S. refinement available in SHELXL, CRYSTALS etc.
- Problem: Determination of the Twin Law that is in effect.
- Partial solution: coset decomposition, try all possibilities (i.e. all symmetry operations of the lattice but not of the structure)
- **ROTAX** (S.Parson et al. (2002) J. Appl. Cryst., 35, 168. (Based on the analysis of poorly fitting reflections of the type  $F(\text{obs}) \gg F(\text{calc})$ )
- **TwinRotMat** Automatic Twinning Analysis as implemented in PLATON (Based on a similar analysis but implemented differently)

TwinRotMat				TwinRotMat MENU	
Analysis of Fo/Fc Data for Unaccounted (Non)Merohedral Twinning for: twln				RefSelMin	
Cell: 0.71079 20.983 20.983 7.644 90.00 90.00 120.00 Spgr: P-3				Delta/Sig	
Criteria: DeltaI/SigmaI .6T, 16.0, DeltaTheta 0.10 Deg., NeelMin = 50				MaxIndexMin	
N(refl) = 4445, N(selected) = 50, IndMax = 25, CrLti = 0.3, CrLti = 0.10				DeltaTheta	
				FullListing	
				EPS-TwinLaw	
				DepTwinMat1	
				DepTwinMat2	
				DepTwinMat3	
				DepTwinMat4	
				EPS-TwinLaw	
				Resolution	
				Zone-H,K,L	
				Up Down	
				Acceptation	
				SelectMat1	
				SelectMat2	
				SelectMat3	
				SelectMat4	
				Half-Crit1	
				Half-Crit2	
				Half-Crit3	
				Half-Crit4	
				End	
				Exit	
				Recursive	

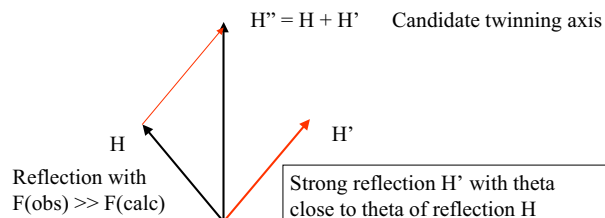
  

PLATON-Eng	6 17:21:12 2005	(60005)	
2-axle ( 0 0 1 ) ( 0 0 1 ), Angle ( ) = 0.00 Deg, Freq = 47			1
(-1.000 0.000 0.000) * (h1) = (h2)	Nr Overlap = 4445		
( 0.000 -1.000 0.000) * (h1) = (h2)	BRSF = 0.54		
( 0.000 0.000 1.000) (L1) = (L2)	DEL-R = -0.107		
2-axle ( 1 -1 0 ) ( 1 -1 0 ), Angle ( ) = 0.00 Deg, Freq = 48			2
( 0.000 -1.000 0.000) (h1) = (h2)	Nr Overlap = 4445		
(-1.000 0.000 0.000) * (h1) = (h2)	BRSF = 0.01		
( 0.000 0.000 -1.000) (L1) = (L2)	DEL-R = -0.001		
2-axle ( 2 -1 0 ) ( 1 0 0 ), Angle ( ) = 0.00 Deg, Freq = 36			3
( 1.000 0.000 0.000) (h1) = (h2)	Nr Overlap = 4445		
(-1.000 -1.000 0.000) * (h1) = (h2)	BRSF = 0.01		
( 0.000 0.000 -1.000) (L1) = (L2)	DEL-R = -0.001		
2-axle ( 1 3 -1 ) ( 10 14 -23 ), Angle ( ) = 0.45 Deg, Freq = 10			4
(-0.732 0.375 -0.608) (h1) = (h2)	Nr Overlap = 576		
( 0.804 0.126 -1.818) * (h1) = (h2)	BRSF = 0.02		
(-0.288 -0.375 -0.394) (L1) = (L2)	DEL-R = 0.000		
twln	R = 0.20		

## Ideas behind the Algorithm

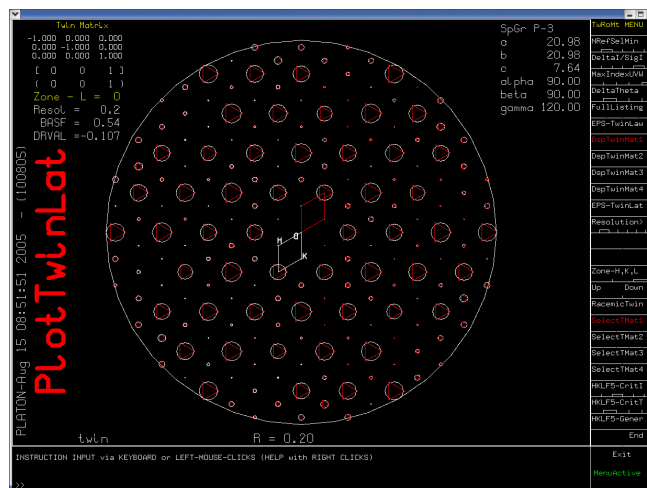
- Reflections effected by twinning show-up in the least-squares refinement with  $F(\text{obs}) \gg F(\text{calc})$
- Overlapping reflections necessarily have the same theta within a tolerance.
- The more interesting cases of twinning in the current context are those with layers of overlapping reflections that can be described with a rotation about a reciprocal axis.

## Possible Twin Axis



## TwinRotMat Algorithm

- Select the set of reflections  $H$  with  $F(\text{obs}) \gg F(\text{calc})$
- Loop over all reflections  $H'$  (including symmetry related ones) for which  $F(H') > F(H)$  and  $\Theta(H') \sim \Theta(H)$ .
- Register  $H'' = H + H'$  (reduced to co-prime integers) as a possible twinning axis (I.e. count the frequency of occurrence)
- Eliminate symmetry directions and  $H''$  that are related by symmetry.
- Determine the twinning factor that gives the lowest R-factor (simple gridsearch).



## Special Implementations

Older programs with dated input.

**StructureTidy** (standardisation of Inorganic Structures). CIF interface generates the proper input in original input format.

## Bond Valence Analysis

## System S

- Automatic structure determination  
(Space group determination, solution, refinement, analysis)
- Build-in in PLATON (Unix only)
- Calls external programs including itself for various functions.
- Program runs in either *guided* or *no-questions-asked* mode

## Concluding Remarks

- The ‘Single Source’ approach of PLATON makes it easy (for me) to implement new tools within the existing framework of already available tools.
- Only one program has to be maintained.
- A one-person project, so no internal discussions.
- Of-course, the above is controversial ...

## Thanks

- Thanks to the users for their:
- Complaints
- Bug reports (‘undocumented features ..)
- Suggestions

# Simple algorithms for macromolecular phasing

IUCr Computing School, Siena,  
August 2005

George M. Sheldrick

<http://shelx.uni-ac.gwdg.de/SHELX/>

## SAD as a special case of MAD

$$|F_+|^2 = |F_T|^2 + a|F_A|^2 + b|F_T||F_A|\cos\alpha + c|F_T||F_A|\sin\alpha$$

$$|F_-|^2 = |F_T|^2 + a|F_A|^2 + b|F_T||F_A|\cos\alpha - c|F_T||F_A|\sin\alpha$$

where  $a = (f''^2 + f'^2)/f_0^2$ ,  $b = 2f'/f_0$ ,  $c = 2f''/f_0$  and  $\alpha = \phi_T - \phi_A$

By subtracting the second equation from the first we obtain:

$$|F_+|^2 - |F_-|^2 = 2c|F_T||F_A|\sin\alpha$$

If we assume that the native structure factor  $|F_T|$  is given by  $|F_T| = 1/2(|F_+| + |F_-|)$ , this simplifies to:

$$|F_+| - |F_-| = c|F_A|\sin\alpha$$

where  $|F_A|$  is the heavy atom structure factor) and  $\phi_T = \phi_A + \alpha$ . Amazingly, this is sufficient to find the heavy atoms and to use them to estimate the protein phases  $\phi_T$  for some reflections.

## Substructure solution

The same methods used for *ab initio* all-atom structure solution from very high resolution native data turn out to be eminently suitable for the location of heavy atom sites from SIR, SAD  $\Delta F$  or MAD  $F_A$  values. The resolution is then not so critical; 3.5Å is fine because it is normally still greater than the distance between the sites. In the case of sulfur-SAD, the two sulfurs in a disulfide bridge fuse into a single super-sulfur atom at this resolution.

The  $\Delta F$  or  $F_A$  values are normalized to give  $E$ -values. The fact that direct methods use only the larger  $E$ -values is an advantage, especially for SIR or SAD, because the  $\Delta F$  values represent lower limits on the heavy atom structure factors and so the weak  $\Delta F$  are very unreliable anyway.

## Experimental phasing of macromolecules

Except in relatively rare cases where atomic resolution data permit the phase problem to be solved by *ab initio* direct methods, experimental phasing usually implies the presence of heavy atoms to provide reference phases. We then calculate the phases  $\phi_T$  of the full structure by:

$$\phi_T = \phi_A + \alpha$$

Where  $\phi_A$  is the calculated phase of the heavy atom substructure. As we will see,  $\alpha$  can be estimated from the experimental data. The phase determination requires the following stages:

1. Location of the heavy atoms.
2. (Refinement of heavy atom parameters and) calculation of  $\phi_A$ .
3. Calculation of starting protein phases using  $\phi_T = \phi_A + \alpha$ .
4. Improvement of these phases by density modification (and where appropriate NCS averaging).

## SAD, SIR, SIRAS and MAD

For SAD, the reflections with the largest normalized anomalous differences  $|E_A|$  will tend to have  $\alpha$  close to 90 or 270°. These reflections are used to find the heavy atoms (only the largest  $|E_A|$  are used by direct methods) and to start the phasing.

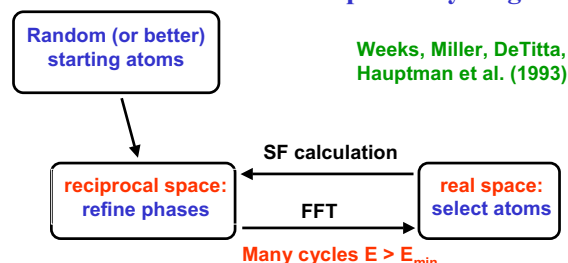
In the case of SIR, if we assume that the isomorphous difference is small compared to the native structure factor, we obtain the approximation:

$$|F_{\text{deriv}}| - |F_{\text{nat}}| = b|F_A|\cos\alpha$$

So reflections with large normalized isomorphous differences will tend to have  $\alpha$  close to 0 or 180°. Although  $||F_{\text{deriv}}| - |F_{\text{nat}}||$  will in general be larger than  $||F_+| - |F_-||$ , as we shall see  $\alpha$  values of 0° or 180° are less useful than 90° or 270°, and there are problems with lack of isomorphism and scaling.

For MAD (and SIRAS) we have  $F_A\sin\alpha$  and  $F_A\cos\alpha$  and so we can derive both  $|F_A|$  and  $\alpha$ .

## Dual space recycling



If the figures of merit indicate a solution, it can be expanded to the complete structure using all data

Implemented in SnB and (later) SHELXD

### Random atom positions

```
JS=MOD(JS*3877+29573,139968)
JK=MOD(JK*3613+45289,214326)
JR=MOD(JR*1366+150889,714025)
X=7.1444902E-6*REAL(JS)
Y=4.6657895E-6*REAL(JK)
Z=1.4005112E-6*REAL(JR)
```

This Fortran code generates random coordinates x, y and z in the range 0...1. The use of three independent series ensures that the repeat length is long ( $3.5 \times 10^{15}$ ). JS etc. can be given fixed starting values so that the same sequence is always generated (with luck also on different computers) or they can be made randomly random (e.g. by using the last few digits of the current time (expressed as a real number in seconds)).

### Selecting vectors for the translational search

```
C
C Choose biased random starting vector (PATS +n)
C
      NJ=LM-JW+1
      IF (NJ.LE.0) GOTO 18
      N=NJ
      DO 20 NW=1,JQ
        JR=MOD(JR*1366+150889,714025)
        N=MAX0(N,LM-MOD(JR,NJ))
20      CONTINUE
```

This SHELXD Fortran code was cited by Ralf Grosse-Kunstleve as a typically cryptic piece of SHELX code (he diplomatically said that he found the comment useful). The general Patterson peaks are stored in XA(JW...LM) etc., JR is a random number and JQ is a 'bias factor' (third PATS parameter, usually 5) that causes (higher) peaks closer to LM to be chosen more often.

### How to find the independent Patterson Vectors

Assume that we wish to find all unique non-origin vectors involving atoms  $x_1, y_1, z_1$  and  $x_2, y_2, z_2$  and their symmetry equivalents in order to calculate a Patterson superposition function and that there are  $N_S$  symmetry operations. Lattice operators are ignored because they will generate equivalent peaks (the Patterson has the same lattice type as the structure) but a center of symmetry should be included in the symmetry operators.

There are  $(N_S - 1)$  unique Harker vectors for each of the two atoms. To find the unique cross-vectors we need to subtract  $x_1, y_1, z_1$  from  $x_2, y_2, z_2$  and all its symmetry equivalents. So the total number of unique vectors generated by a two-atom search fragment is  $2(N_S - 1) + N_S$ . In the space group  $P4_32_12$  this is 22. For an  $N_A$ -atom fragment the number is  $N_A(N_S - 1) + \frac{1}{2}(N_A - 1)N_A N_S$ .

Note that in high-symmetry cases some Harker vectors have multiplicities greater than one, and that in centrosymmetric space groups all non-Harker have multiplicities of at least two.

### Probabilistic Patterson sampling (PATS in SHELXD)

Each unique general Patterson vector of suitable length is a potential HA-HA vector, and may be employed as a 2-atom search fragment in a translational search based on the *Patterson minimum function*. Alternatively a vector of known length – e.g. a S-S bond (2.03Å) – but random orientation can be used. For each position of the two atoms in the cell, the Patterson height  $P_j$  is found for all vectors between them and their equivalents, and the sum (PSUM) of the lowest (say) 35% of  $P_j$  calculated.

It would be easy to find the global maximum of PSUM using a fine 3D grid, but this often does NOT lead to the solution of the structure! A more effective approach is to generate many different starting positions by simply taking the best of a finite number of random trials each time.

The *full-symmetry Patterson superposition minimum function* is used to expand from the two atoms to a much larger number before entering the dual-space recycling.

### Calculating the Patterson minimum function

In its simplest form, the Patterson minimum function is simply the lowest Patterson density at a series of points in the Patterson, e.g. in the case of the symmetry minimum function these are all vectors between one site and its symmetry equivalents.

When more than one site is involved or the symmetry is high, the chance of an accidental low value is too high, so Schilling (1970) and Nordman (1980) improved the function by summing the lowest (say) 1/3 of the Patterson densities involved. Since this requires sorting the values, it can become rate determining.

Alternatively one can sum some suitable function of the Patterson densities  $\rho$  designed to put more weight on the lower values. Summing  $s\sqrt{|\rho|}$  where  $s$  is the sign of  $\rho$  works quite well, even in high symmetry space groups and with noisy data, which can cause problems for the Schilling/Nordman method.

### The full-symmetry Patterson superposition minimum function

SHELXD finds good (*but different*) positions for a two-atom fragment by trying (say) 10000 random translations and using the PSMF as a criterion. The *full-symmetry PSMF* is then calculated one pixel at a time. A dummy atom is placed on the pixel and the Patterson function values at all vectors involving it, the two atoms of the search fragment, and their symmetry equivalents found. The sum of the lowest (say 1/3) of these values (the PSMF) is stored at that pixel.

The resulting map is then peak-searched to find the starting atom sites in the dual-space recycling part of the SHELXD procedure for finding the heavy atom sites. Each overall trial generates a *different* starting set of sites that are relatively consistent with the Patterson. There is no limit to the number of starting sets that can be generated in this way.

## Density modification

The heavy atoms can be used to calculate reference phases; initial estimates of the protein phases can then be obtained by adding the phase shifts  $\alpha$  to the heavy atom phases as explained at the beginning of this talk.

These phases are then improved by density modification. Clearly, if we simply do an inverse Fourier transform of the unmodified density we get back the phases we put in. So we try to make a 'chemically sensible' modification to the density before doing the inverse FFT in the hope that this will lead to improved estimates for the phases.

Many such density modifications have been tried, some of them very sophisticated. Major contributions have been made by Kevin Cowtan and Tom Terwilliger. One of the simplest ideas, truncating negative density to zero, is actually not too bad (it is the basic idea behind the program ACORN).

## The sphere of influence algorithm

The variance  $V$  of the density on a spherical surface of radius 2.42 Å is calculated for each pixel in the map. The pixels with the highest  $V$  are most likely to correspond to real protein atomic positions.

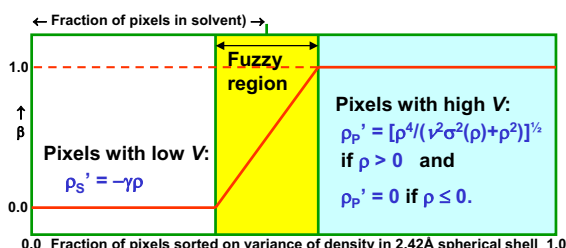
Pixels with low  $V$  are *flipped* ( $\rho_s' = -\gamma\rho$  where  $\gamma$  is about one).

For pixels with high  $V$ ,  $\rho$  is replaced by  $[\rho^4/(v^2\sigma^2(\rho)+\rho^2)]^{1/2}$  (with  $v$  usually 1.0) if positive and by zero if negative. This has a similar effect to the procedure used in the program ACORN.

A *fuzzy boundary* is used; in the *fuzzy region*  $\rho$  is set to a weighted sum of the two treatments. The *fuzzy boundary* is an attempt to avoid the *lock-in effect* of a binary mask.

The use of a spherical surface rather than a spherical volume was intended to add a little chemical information (2.42 Å is a typical 1,3-distance in proteins and DNA). An empirical weighting scheme for phase recombination is used to combat model bias.

## The fuzzy solvent boundary

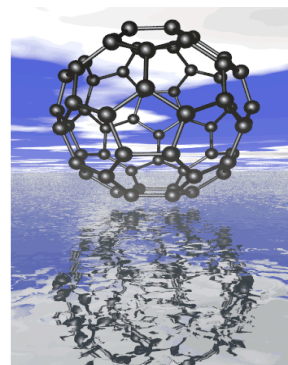


In the *fuzzy region*, the modified density is a weighted mean of the two treatments:  $\rho' = \beta\rho_p' + (1-\beta)\rho_s'$

The parameters  $\gamma$  and  $v$  are both usually set to 1.0

## Calculating the sphere of influence

In SHELXE, the following method is used to generate the sphere. A  $C_{60}$  molecule consists of five and six-membered rings. If we make a vector from the center to each atom and also from the center to the center of each of the 32 faces, we define 92 directions that are well distributed in space. These 92 directions are stored in the form of 92 triplets of pixel offsets with vector lengths close to 2.42 Å. These are added to the coordinates of each pixel in turn to calculate the variance of the density in the *sphere of influence* of each pixel.



Graphic by Voita Jancik

## The free lunch algorithm

In two recent papers (Caliandro *et al.*, Acta Cryst. D61 (2005) 556-565 and 1080-1087) the Bari group around Carmelo Giacovazzo used density modification to calculate phases for reflections that had not been measured, either completing the data to a given resolution or extending the resolution.

Their unexpected conclusion was that if these phases are now used to recalculate the density, using very rough estimates of the (unmeasured) amplitudes, the density actually improves! I have incorporated this into a test version of SHELXE and can completely confirm their observations, at least when the native data have been measured to a resolution of 2 Å or better.

Since one is apparently getting something for nothing, I propose that this algorithm be named the *free lunch algorithm*.

## Solution of an unknown structure

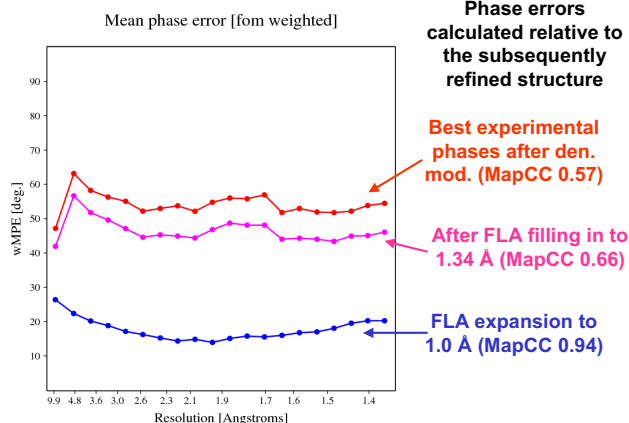
The free lunch algorithm (FLA) clearly improved the density for a number of standard test structures, introducing real features that were not present in the original maps. However a particularly convincing example was the application to the solution of an unsolved structure by Isabel Usón using data collected by Clare Stevenson.

Data for this 262 amino-acid protein in space group P2 were almost complete to 1.9 Å and somewhat partial to 1.35 Å. Despite collecting six datasets the only phase information was a weak SIRAS signal to about 3.5 Å from a mercury acetate derivative. All attempts to improve these maps by interpretation or modification of the density and also molecular replacement on the native data failed.

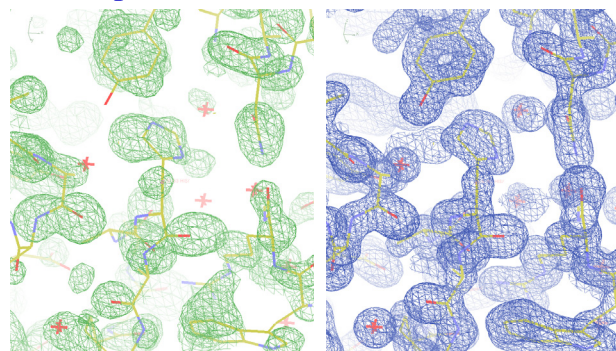
At first we thought that the free lunch algorithm might be able to fill in the missing data, but Isabel was ambitious and expanded to 1.0 Å, much further than the crystals had ever diffracted.



## Postmortem on a free lunch



## Maps before and after a free lunch



Best experimental phases after den. mod. (MapCC 0.57)

After expansion to 1.0 Å with virtual data (MapCC 0.94)

## Why do we get a free lunch?

It is not immediately obvious why inventing extra data improves the maps. Possible explanations are:

1. The algorithm corrects Fourier truncation errors that may have had a more serious effect on the maps than we had realised.
2. Phases are more important than amplitudes (see Kevin's ducks and cats!), so as long as the extrapolated phases are OK any amplitudes will do.
3. Zero is a very poor estimate of the amplitude of a reflection that we did not measure.

## Acknowledgements

I am particularly grateful to Isabel Usón, Thomas R. Schneider, Stephan Rühl and Tim Grüne for many discussions.

**SHELXD:** Usón & Sheldrick (1999), *Curr. Opin. Struct. Biol.* 9, 643-648; Sheldrick, Hauptman, Weeks, Miller & Usón (2001), *International Tables for Crystallography Vol. F*, eds. Arnold & Rossmann, pp. 333-351; Schneider & Sheldrick (2002), *Acta Cryst. D* 58, 1772-1779.

**SHELXE:** Sheldrick (2002), *Z. Kristallogr.* 217, 644-650; Debreczeni, Bunkóczi, Girmann & Sheldrick (2003), *Acta Cryst. D* 59, 393-395; Debreczeni, Bunkóczi, Ma, Blaser & Sheldrick (2003), *Acta Cryst. D* 59, 688-696; Debreczeni, Girmann, Zeeck, Krätzner & Sheldrick (2003), *Acta Cryst. D* 59, 2125-2132.

<http://shelx.uni-ac.gwdg.de/SHELX/>



## Automation of structure determination

*Use of scoring procedures to assist in decision-making*

*Simple procedures for automation choosing the current best path at each decision-point*

## What is automation?

*Procedures (things to do)*

*Control (deciding what to do)*

## What is automation?

*Automation as a set of linked procedures*

*Each procedure has clearly-defined...*

*Inputs  
Methods to apply to inputs  
Outputs*

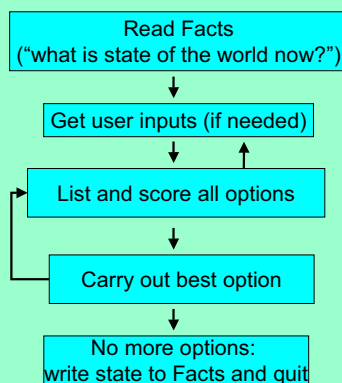
## What is automation?

*Automation as a set of linked procedures*

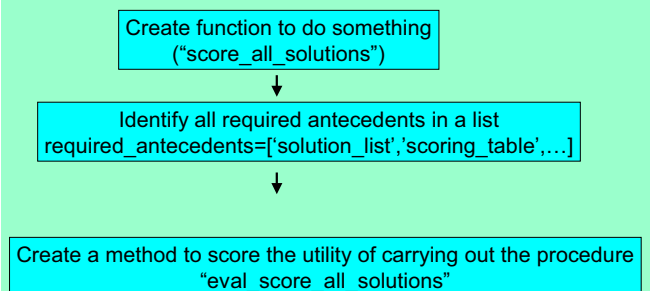
*Control steps have clearly-defined...*

*Possible decisions to make  
Information required to make decisions  
Next step(s) to take based on decisions  
(Including...what to do if things go wrong)*

*Simple automation using a scoring scheme for decision-making  
(as implemented in PHENIX wizards)*



## Modular PYTHON routines in AutoSol



### Deciding which solutions to follow up: "COVERAGE"

User sets "coverage" = "the desired confidence of keeping the best solution in consideration"

Score solutions, with confidence intervals

Follow up on any solution that could really be the top one (i.e., top solution Z-score = 14, next solution Z=13, "coverage"=95% -> carry through with BOTH solutions because either could be the best)

### Automation of structure determination

Scale data

HYSS heavy-atom search

Score and rank solutions

Phase and quick density modification  
Get sites with difference Fourier

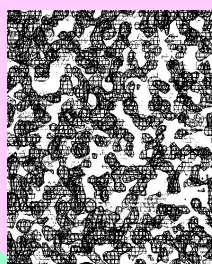
Coverage satisfactory:  
go on to full density modification and iterative model-building

**PHENIX AutoSol wizard standard sequence**

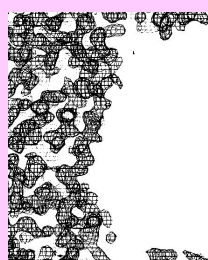
### Why we need good measures of the quality of an electron-density map:

Which solution is best?

Are we on the right track?



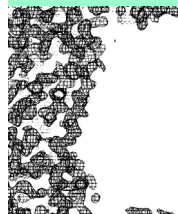
If map is good:  
It is easy  
(which is correct?)



### Evaluating electron density maps: Methods examining the map itself

Basis	Good map	Random map
Skew of density (Podjarny, 1977)	Highly skewed (very positive at positions of atoms, zero elsewhere)	Gaussian histogram
SD of local rms densities (Terwilliger, 1999)	Solvent and protein regions have very different rms densities	Map is uniformly noisy
Connectivity of regions of high density (Baker, Krukowski, & Agard, 1993)	A few connected regions can trace entire molecule	Many very short connected regions
Presence of tubes of density or helices/strands or local patterns in map (Colovos, Toth & Yeates, 2001; Terwilliger, 2004)	CC of map with a filtered version is high	CC with filtered version is low

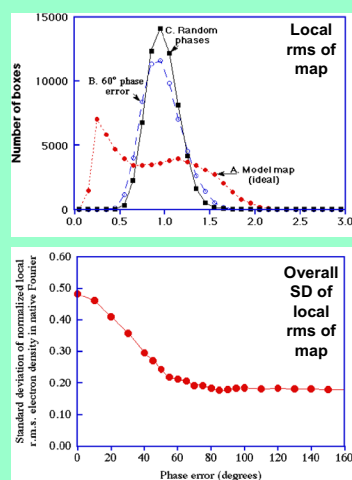
### Scoring: does the native Fourier look like a protein?



A good map:  
clear solvent  
vs protein



A poor map:  
it looks the same  
all over



### Evaluating electron density maps: Methods examining the map itself

Basis	Good map	Random map
Skew of density (Podjarny, 1977)	Highly skewed (very positive at positions of atoms, zero elsewhere)	Gaussian histogram
SD of local rms densities (Terwilliger, 1999)	Solvent and protein regions have very different rms densities	Map is uniformly noisy
Connectivity of regions of high density (Baker, Krukowski, & Agard, 1993)	A few connected regions can trace entire molecule	Many very short connected regions
Presence of tubes of density or helices/strands or local patterns in map (Colovos, Toth & Yeates, 2001; Terwilliger, 2004)	CC of map with a filtered version is high	CC with filtered version is low

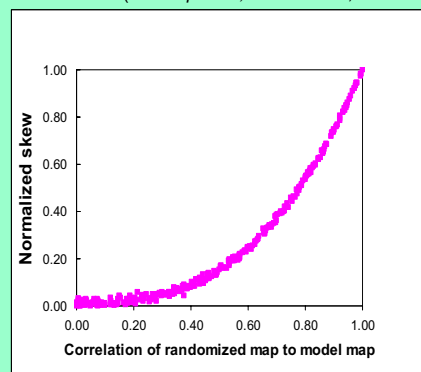
## Evaluating electron density maps

Methods based on density-modification and R-factors

Basis	Good map	Random map
R-factor in 1 <sup>st</sup> cycle of density modification (Cowtan, 1996)	Low R-factor	High R-factor
Correlation of map made with map-probability phases with original map (Terwilliger, 2001)	High correlation	Lower correlation
(map-probability from solvent flattening or from truncation at high density level)		

## Skew of electron density in maps of varying quality

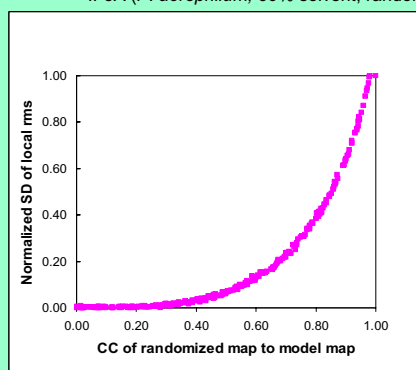
IF5A (*P. aerophilum*, 60% solvent; randomized maps)



(High electron density at positions of atoms; near zero everywhere else => high skew for good map)

## SD of local rms of electron density in maps of varying quality

IF5A (*P. aerophilum*, 60% solvent; randomized maps)

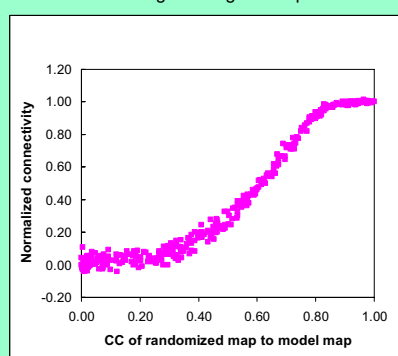


(Large rms in protein region; low in solvent => high SD for good map)

## Connectivity of maps of varying quality

IF5A (*P. aerophilum*, 60% solvent; randomized maps;

Number of contiguous regions required to enclose top 5% of density)



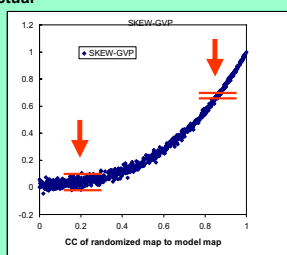
(Most of high density is connected in a good map)

## Bayesian estimation of map quality from skew measurement on map

Start with database of randomized model data:

What values of skew do I measure if the actual map correlation is CC?

$$CC \rightarrow P(\text{skew}_{\text{obs}} | CC)$$



## Bayesian estimation of map quality from skew measurement on map

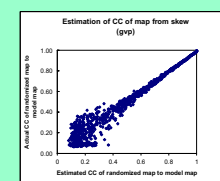
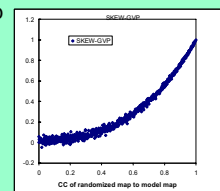
Given measurement of skew : estimate CC...

For each possible value of CC:

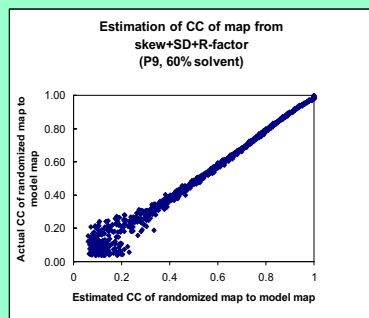
"probability that CC is correct is proportional to probability of measuring skew<sub>obs</sub> given this CC"

$$P(CC) = \alpha P(\text{skew}_{\text{obs}} | CC)$$

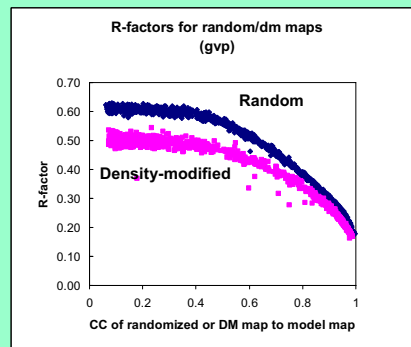
Combine all independent sources of information



Bayesian estimation of map quality using Skew, SD of local rms density, R-factor



R-factors for density-modified maps are systematically lower than those of randomized maps of same quality

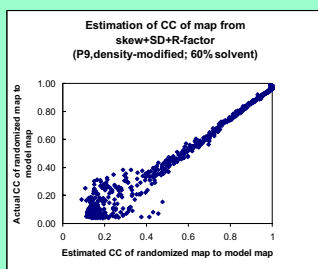
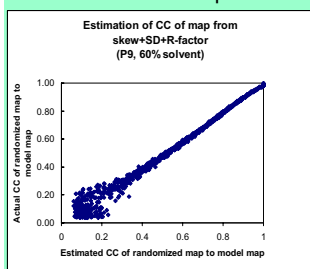


Bayesian estimation of map quality

Estimates for randomized maps are much better than those of density-modified maps

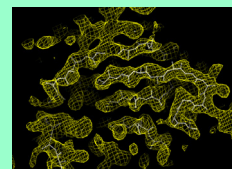
Randomized maps

Density-modified maps



*Model-building at moderate resolution using scoring methods for decision-making*

(following ideas of T.A. Jones, Cowtan, Oldfield, McRee, Levitt, Perrakis, Lamzin)

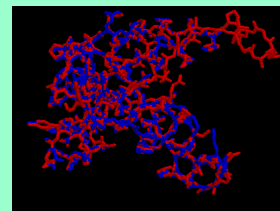


•FFT-based identification of helices and strands

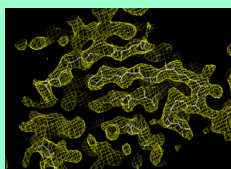
•Extension with tripeptide libraries

•Probabilistic sequence alignment

•Automatic molecular assembly

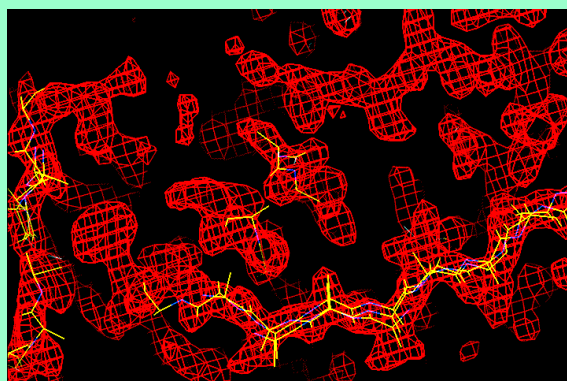


*Placement of helical and extended templates*

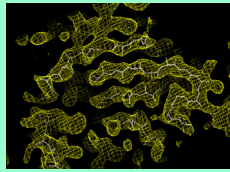


- Identify locations with FFT-based convolution search
- Maximize CC of template with map
- Superimpose each fragment in corresponding library (helix, sheet) on template
- Identify longest segment in good density, score =  $\langle \text{density} \rangle \times \sqrt{\text{Natoms}}$

*Initial model-building – strand fragments*

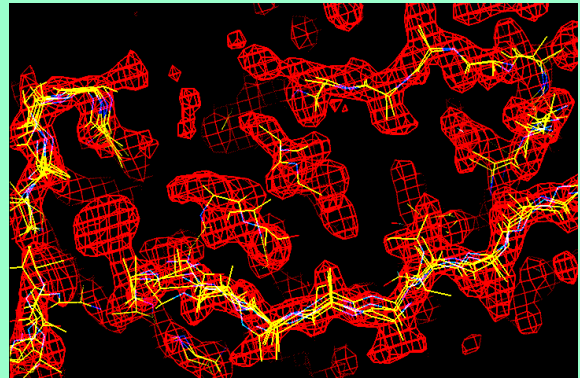


### Chain extension by placement of tripeptide fragments

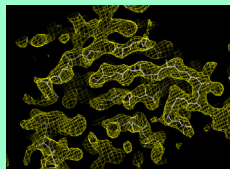


- Look-ahead scoring: find fragment that can itself be optimally extended
- C-terminal extension. Start at C-terminus of protein
- Each of 10000 fragments: superimpose CA C O on same atoms of last residue in chain (extending by 2 residues): pick best 10
- Each of best 10: extend again by 2 residues and pick best 1; score for 2-residue extension= best <density> for 4-residue extension based on this 2-residue extension
- N-terminal: same, but going in opposite direction

### Chain extension with tripeptide libraries (result: many overlapping fragments)

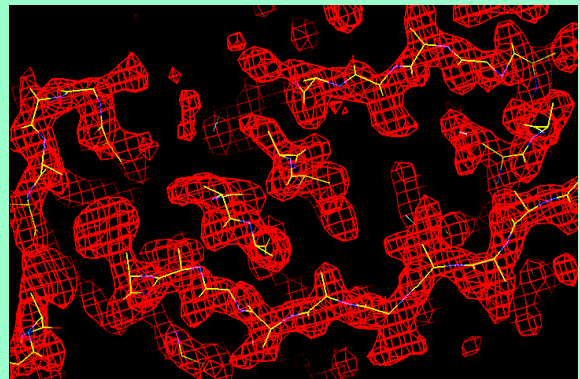


### Assembly of main-chain

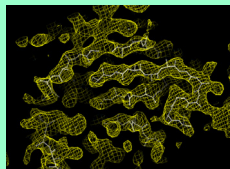


- Choose highest-scoring fragment
- Test all overlapping fragments as possible extensions
- Choose one that maximizes score when put together with current fragment
- When current fragment cannot be extended: remove all overlapping fragments, choose best remaining one, and repeat

### Main-chain as a series of fragments (choosing the best fragment at each location)



### Scoring side-chain templates at each position

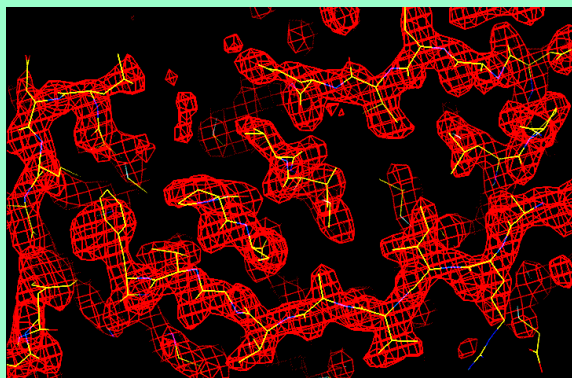


- Identify side-chain orientation from N C A C of main-chain
- Get CC of template with density -> Z-score
- (Compare CC with mean, SD of all side chain density with this template)
- $P(\text{this side-chain/rotamer is correct}) = P_o(\text{this side-chain/rotamer}) \cdot P(Z)$

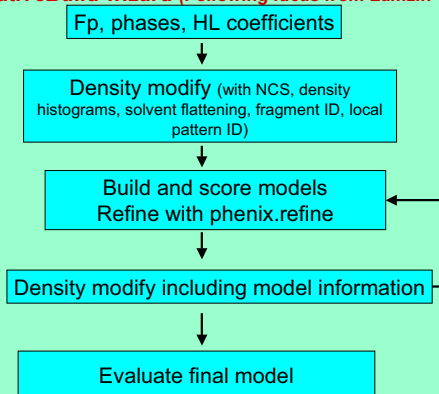
### Side-chain template matching to identify sequence alignment to map (IF5A data) Relative probability for each amino acid at each position (Correct amino acids in bold)

#	G	A	S	V	I	L	M	C	F	Y	K	R	W	H	E	D	Q	N	P	T
1	6	5	4	18	<b>18</b>	6	1	1	1	2	6	2	2	1	9	6	1	0	1	4
2	4	11	14	<b>37</b>	5	2	0	2	0	0	2	3	0	0	1	2	0	0	0	6
3	11	<b>23</b>	5	12	5	3	2	0	1	3	7	3	1	0	5	3	2	0	2	2
4	7	9	6	<b>16</b>	8	5	2	0	1	3	8	4	1	0	7	6	2	0	3	4
5	<b>31</b>	7	3	7	4	2	1	0	1	3	5	4	1	0	6	2	2	0	11	1
6	1	3	3	<b>41</b>	14	8	0	0	0	0	2	1	0	0	2	4	0	0	1	9
7	0	0	0	0	0	0	0	0	<b>15</b>	<b>63</b>	1	0	17	1	0	0	0	0	0	0
8	2	3	6	<b>23</b>	10	6	2	1	0	1	4	3	0	0	5	<b>16</b>	1	0	1	6
9	<b>96</b>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

### Addition of side-chains to fixed main-chain positions

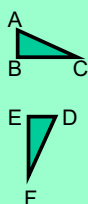


### Iterative model-building, density modification and refinement at moderate resolution using the PHENIX IterativeBuild wizard (Following ideas from Lamzin & Perrakis)

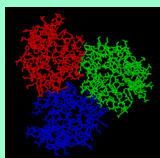


### Automated NCS identification from heavy-atom sites

- Expand heavy-atom sites within radius R of origin
- Make list of all pairs of sites, sorted by distance between sites  $d$
- Choose any 3 HA sites – a triangle ABC
- Find all other sets of 3 HA sites that form the same triangle
  - If some exist (DEF) -> this might correspond to NCS
  - If none...try another set of 3 HA sites



- Testing NCS: Sites ABC match sites DEF
- Does density near ABC match (after rotation/translation) density near DEF?



### Automated NCS identification using heavy-atom sites and analysis of the electron-density map

Structure	Number of sites	NCS	NCS (found from heavy-atom sites)	NCS (electron-density map)
NDP Kinase	9	3-fold	3-fold	3-fold
Hypothetical	16	2-fold	2-fold	2-fold
Red Fluorescent Protein	26	4 copies	4 copies	4 copies
AEP Transaminase	66	6 copies	6 copies	6 copies
Formate dehydrogenase	12	2-fold	2-fold*	2-fold
Gene 5 protein	2	None	None	None
Armadillo repeat from $\beta$ -catenin	15	None	2 copies	None
Dehalogenase	13	None	3 copies	None
Initiation Factor 5A	4	None	None	None

### Molecular assembly in RESOLVE

List all chains assigned to sequence (anywhere in space)

A possible arrangement consists of:

- Each chain assigned to a molecule
- Each chain assigned to a symmetry-related position

Score a possible arrangement based on:

- Plausibility of gap distances between position of C of residue i and N of residue j
- RMS distance of chains from molecular center
- RMSD of NCS symmetry for corresponding atoms

- Try a reasonable starting arrangement (each chain assigned to the center of an NCS copy)
- Adjust by moving chains and groups of chains randomly from one symmetry-related position to another. Choose based on score.

### Molecular assembly in RESOLVE

Summary of molecular assembly results (NDP-kinase)

NCS copies: 3

Molecule: 1 Chain: 1 Score for molecular location: 0.83

Frag	Start	End	N	Overlap	Link Length	Mol Radius	NCS RMSD	NCS <N>	Score
1	17	64	48	0	6.6	4.5	0.7	31.0	51.0
2	69	74	6	0	24.5	19.6	0.5	3.0	3.7
3	115	137	23	0	14.4	5.2	0.8	20.5	22.7
4	166	186	21	0		5.2	0.6	9.5	22.4

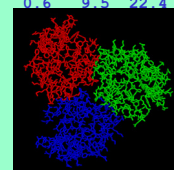
Residues placed for this molecule: 98

Total residues placed: 309 of 588 or 52%

Residues built without side chains: 65

Total residues built: 374 or 63%

Total score for this arrangement: 314.4



## Automation of structure determination

*Use of scoring procedures to assist in decision-making*

*Simple procedures for automation choosing the current best path at each decision-point*

## The *PHENIX* project



Crystallographic software for automated structure determination

### Computational Crystallography Initiative (LBNL)

-Paul Adams, Ralf Grosse-Kunstleve, Peter Zwart, Nigel Moriarty, Nicholas Sauter, Pavel Afonine



### Los Alamos National Lab (LANL)

-Tom Terwilliger, Li-Wei Hung, Thiru Radhakannan



### Cambridge University

-Randy Read, Airlie McCoy, Laurent Storoni, Hamsapriye



### Texas A&M University

-Tom Ioerger, Jim Sacchettini, Kreshna Gopal, Lalji Kanbi, Erik McKee, Tod Romo, Reetal Pai, Kevin Childs, Vinod Reddy



## Acknowledgements

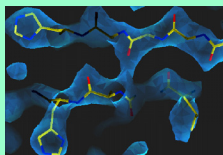
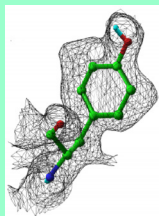
**PHENIX:** [www.phenix-online.org](http://www.phenix-online.org)

Paul Adams, Ralf Grosse-Kunstleve, Nigel Moriarty, Nick Sauter, Pavel Afonine, Peter Zwart (LBNL Computational Crystallography Initiative)

Randy Read, Airlie McCoy, Laurent Storoni, Hamsapriye (Cambridge)

Tom Ioerger, Jim Sacchettini, Kreshna Gopal, Lalji Kanbi, Erik McKee, Tod Romo, Reetal Pai, Kevin Childs, Vinod Reddy (Texas A&M)

Li-wei Hung, Thiru Radhakannan (Los Alamos)



Generous support for PHENIX from the NIGMS Protein Structure Initiative



## Crystallographic Symmetry

### Crystallographic Symmetry in Real and Reciprocal Space.

Kevin Cowtan  
cowtan@ysbl.york.ac.uk

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Crystal Symmetry

## Crystallographic symmetry

### Crystallographic symmetry in Real Space

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Crystal Symmetry

## Crystallographic symmetry:

### Overview:

- Brief review of concepts.
- Spacegroups and settings.
  - Spacegroup symbols.
- Symmetry operators.
- Symmetry in reciprocal space.
- Symmetry in real space.
- Ideas and implementations.

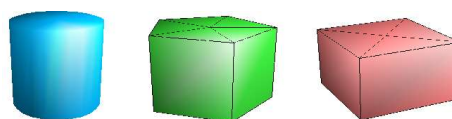
Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Crystal Symmetry

## Crystallographic symmetry:

### Concepts:

- Point group – symmetries about a point.
  - e.g. n-fold rotation (any n), mirror, inversion.



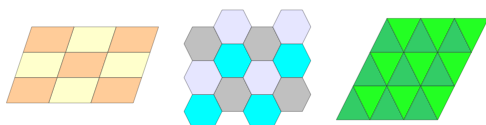
Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Crystal Symmetry

## Crystallographic symmetry:

### Concepts:

- Lattice group – symmetries of a lattice.
  - Lattice can be made of any shape which tessellates.
    - 2D: square, rectangle, rhombus, parallelogram, triangle, hexagon.
    - 3D: prisms of above, tetrahedron, parallelepiped.
  - Lattice symmetries restricted by those shapes.



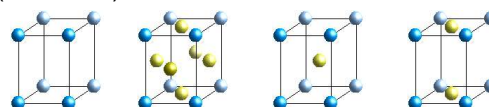
Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Crystal Symmetry

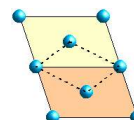
## Crystallographic symmetry:

### Concepts:

- Bravais lattice – parallelepiped cell with lattice **centerings** to represent the other shapes. (P/F/I/C/R).



- Primitive cell – Un-centered cell chosen from within a centered lattice.



## Crystallographic symmetry:

Space group:

- Combines symmetry of lattice, lattice centering, and symmetry within the primitive cell.
- 230 distinct space groups (i.e. combinations of symmetries consistent with 3D lattice).
  - Tabulated and numbered in International Tables for Crystallography.
  - Also described by Hermann-Mauguin symbols.
    - e.g. Spacegroup 19 =  $P2_12_12_1$

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry:

- Which is great, but what we really want to know is where the atoms are. Which means knowing the cell and the symmetry operators.
  - e.g. If there is an atom at  $\underline{x} = (13, 17, 24)$ , then  $\underline{u} = (0.3, 0.2, 0.4)$ , and the symop is  $(-u, v+1/2, -w)$ , so there is an atom at  $\underline{u} = (-0.3, 0.7, -0.4)$  or  $\underline{x} = (-13, 45, -24)$
- **PROBLEM:** The space group number (or H-M symbol) do not uniquely determine the symmetry operators.
  - Several space groups have multiple *tabulated* settings.
  - And there are a huge number of possible non-standard settings.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry:

- Solutions:
  - Only allow certain settings.
    - Inconvenient for users and hard to spot. More user support.
  - Use a symbol which includes a precise definition of the setting: Hall symbol from **CCTBX** (also in **Clipper**).
    - Hall & Grosse-Kunstleve (2001) Int Tab B, 201.
  - Ignore the space-group symbols, use the symmetry operators.
    - No ambiguity.
    - Symmetry operators present in CCP4 MTZ/map files, and deposited PDB files.
    - Determining other space-group info from operators complex, but already implemented in **CCTBX**, **Clipper**, **CCP4**.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry: Symops

Symmetry operators (Symops):

- Express the symmetry relationships in the unit cell (and therefore in the diffraction pattern).
- Fractional rotation-translation operators.
- For any spacegroup, let there be  $N_{sym}$  symops, numbered  $0 \dots N_{sym}-1$ 

$$S_i(\underline{u}) = \underline{S}_i \underline{u} + \underline{S}_i$$
- Operator 0 is the identity.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry: Symops

Symmetry operators (Symops):

- We can extract from the  $N_{sym}$  symops, two subgroups of operators: the centering operators and the primitive operators, such that:
 
$$N_{sym} = N_{primitive} \times N_{centering}$$
- The centering operators have  $\underline{S} = \underline{I}$ .
- The primitive operators may have translation parts, but all have  $\underline{S} \neq \underline{I}$ .
  - These are the operators from the corresponding "P" space-group.
  - In reciprocal space we generally ignore the centering operators.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry: Symops

Symmetry operators (Symops):

- e.g. **P 2<sub>1</sub>b** (  $P\ 1\ 2_1\ 1$  , spacegroup 4 )
- Equivalent posns:  $u.v.w: -u.v+1/2,-w;$

$$\underline{S}_0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \underline{S}_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\underline{S}_1 = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \quad \underline{S}_1 = \begin{pmatrix} 0 \\ 1/2 \\ 0 \end{pmatrix}$$

## Crystallographic symmetry: Symops

Symmetry operators (Symops):

- e.g. **P 31** (  $P 3_1$  , spacegroup 144 )
- Equivalent posns:  $u,v,w$ :  $-v,-u-v,w+1/3$ ;  $-u+v,-u,w+2/3$ ;

$$\mathbf{S}_0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{S}_1 = \begin{pmatrix} 0 & -1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{S}_2 = \begin{pmatrix} -1 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{S}_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad \mathbf{S}_1 = \begin{pmatrix} 0 \\ 0 \\ 1/3 \end{pmatrix} \quad \mathbf{S}_2 = \begin{pmatrix} 0 \\ 0 \\ 2/3 \end{pmatrix}$$

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry: Symops

Symmetry operators (Symops):

- e.g. **C 2 2** (  $C 2 2 2$  , spacegroup 21 )

- Equivalent posns:
  - $u,v,w$ ;
  - $-u,-v,w$ ;
  - $-u,v,-w$ ;
  - $u,-v,-w$ ;
  - $u+1/2,v+1/2,w$ ;
  - $-u+1/2,-v+1/2,w$ ;
  - $-u+1/2,v+1/2,-w$ ;
  - $u+1/2,-v+1/2,-w$ ;
- Primitive:
  - $u,v,w$ ;
  - $-u,-v,w$ ;
  - $-u,v,-w$ ;
  - $u,-v,-w$ ;
- Centering:
  - $u,v,w$ ;
  - $u+1/2,v+1/2,w$ ;

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry: Symops

Symmetry operators (Symops):

- Remember: symops are **fractional**, and therefore the matrix is not a true rotation.
- Convert the symop to **orthogonal** form, and it may be used to transform orthogonal coordinates.
  - If the cell is consistent, the matrix part should become a true rotation.
- We can also convert symops to **grid coordinates** by scaling the translations (assuming grid is consistent with symmetry). Useful optimization when handling maps.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry: Symops

Symmetry operators (Symops):

- Related symmetry groups:
  - Point group: (symmetry of anomalous data)
    - Just take the unique rotation parts of the symmetry operators.
  - Laue group: (symmetry of non-anomalous data)
    - Point group + inversion operator.
  - Patterson group: (symmetry of the Patterson map)
    - Laue group + centering operators.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry:

Dealing with symmetry in mathematics:

- In our Likelihood functions we often treat different reflections as independent. But symmetry related reflections (and Friedel opposites) are not independent – these must be handled explicitly.

Dealing with symmetry in software:

- Symmetry related values should never be inconsistent. When we change a structure factor or density, every related value should change immediately.
  - Only store a unique set of values (asymmetric unit), and generate related values on-the-fly.

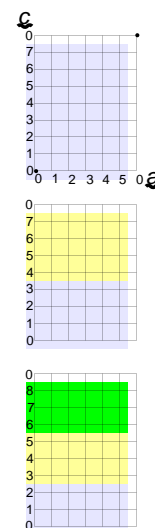
Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry:

Map asymmetric units (ASUs):

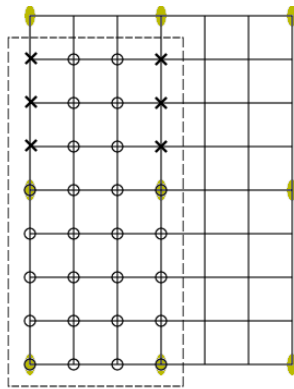
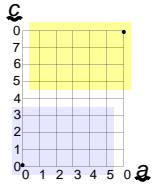
- In **P 1**, the ASU is the whole cell.
- In **P 2<sub>1</sub>y** (  $P 1 2_1 1$  , spacegroup 4 ),
  - Symops are:  $u,v,w$ ;  $-u,v+1/2,-w$ ;
  - Use  $v+1/2$  to generate half the cell along the  $b$  axis.
- For any screw axis, divide the cell length by the screw translation.
  - e.g. **P 31**



## Crystallographic symmetry: Maps

Map asymmetric units (ASUs):

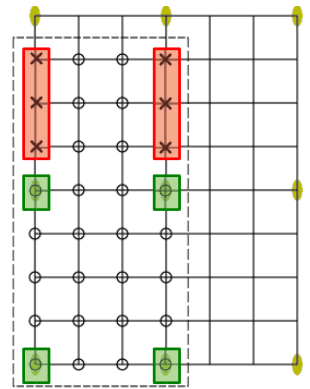
- In **P 2<sub>1</sub>b** ( **P 1 2 1** ),
  - Symops are:  
 $u, v, w; -u, v, -w;$
  - Several sensible ASUs



## Crystallographic symmetry: Maps

Map asymmetric units (ASUs):

- We can define a box which roughly encloses the ASU.
- Some points may still be duplicates.
- Other points may be related to themselves:
  - 'symmetry enhanced'
  - in atom density calculation, they may require special treatment.



## Crystallographic symmetry

Crystallographic symmetry in Reciprocal Space

## Crystallographic symmetry: $hkl$

- Relationships between reflections:

$$\begin{aligned} E(h) &= \sum_j f_j(h) \exp(2\pi i \underline{h}^T \underline{u}_j) \\ &= \sum_j f_j(h) \exp(2\pi i \underline{h}^T [\underline{S}_k \underline{u}_j + \underline{S}_k]) \\ &= \sum_j f_j(h) \exp(2\pi i [\underline{h}^T \underline{S}_k \underline{u}_j + \underline{h}^T \underline{S}_k]) \\ &= \sum_j f_j(h) \exp(2\pi i [(\underline{S}_k^T \underline{h})^T \underline{u}_j + \underline{h}^T \underline{S}_k]) \end{aligned}$$

but:

$$E(\underline{S}_k^T \underline{h}) = \sum_j f_j(h) \exp(2\pi i (\underline{S}_k^T \underline{h})^T \underline{u}_j)$$

therefore:

$$\begin{aligned} E(h) &= E(\underline{S}_k^T \underline{h}) \exp(2\pi i \underline{h}^T \underline{S}_k) \\ E(\underline{S}_k^T \underline{h}) &= E(h) \exp(-2\pi i \underline{h}^T \underline{S}_k) \end{aligned}$$

## Crystallographic symmetry: $hkl$

- Example: **P 3<sub>1</sub>**

$$\begin{aligned} \underline{S}_0 &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} & \underline{S}_0 &= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \\ \underline{S}_1 &= \begin{pmatrix} 0 & -1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} & \underline{S}_1 &= \begin{pmatrix} 0 \\ 0 \\ 1/3 \end{pmatrix} \\ \underline{S}_2 &= \begin{pmatrix} -1 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} & \underline{S}_2 &= \begin{pmatrix} 0 \\ 0 \\ 2/3 \end{pmatrix} \end{aligned}$$

- Symmetry related reflections are:  $\underline{S}_k^T \underline{h}$

$$\begin{pmatrix} h \\ k \\ l \end{pmatrix}, \begin{pmatrix} k \\ -h-k \\ l \end{pmatrix}, \begin{pmatrix} -h-k \\ h \\ l \end{pmatrix}$$

## Crystallographic symmetry: $hkl$

- Relationships between reflections:

$$E(\underline{S}_k^T \underline{h}) = E(h) \exp(-2\pi i \underline{h}^T \underline{S}_k)$$

- But: Sometimes the symmetry operation relates a reflection to itself or its Friedel opposite.  
e.g. (1,0,0) under (u,-v,-w) or (-u,-v,w)

- We know:

$$\begin{aligned} E(h) &= E(h) && \text{(by definition)} \\ E(h) &= E(-h)^* && \text{(Hermitian symmetry)} \end{aligned}$$

## Crystallographic symmetry: $hkl$

- Suppose:  

$$\mathbf{S}_k^T \underline{h} = \underline{h}$$
- Then:  

$$E(\underline{h}) = E(\underline{h}) \exp(-2\pi i \underline{h}^T \underline{S}_k)$$
- This can only be true if  $\underline{h}^T \underline{S}_k$  is an integer.
  - If  $\underline{h}^T \underline{S}_k$  is an integer,  $E(\underline{h})$  is an enhanced reflection, i.e. its intensity  $I(\underline{h})$  is increased by a factor of  $\epsilon$ , where  $\epsilon$  is the number of operators relating  $\underline{h}$  to itself.
  - If  $\underline{h}^T \underline{S}_k$  is not an integer,  $E(\underline{h})$  is a systematic absence.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry: $hkl$

- Suppose:  

$$\mathbf{S}_k^T \underline{h} = -\underline{h}$$
- Then:  

$$E(\underline{h})^* = E(\underline{h}) \exp(-2\pi i \underline{h}^T \underline{S}_k)$$

$$-\phi(\underline{h}) = \phi(\underline{h}) - 2\pi \underline{h}^T \underline{S}_k + 2n\pi$$

$$\phi(\underline{h}) = \pi \underline{h}^T \underline{S}_k + n\pi$$
 i.e. one of two values separated by  $\pi$ .  
 e.g.  $0, \pi$ ;  $+\pi/2, -\pi/2$ ;  $-\pi/3, +2\pi/3$ ;
- The reflection is centric.
  - It may also be enhanced.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry: $hkl$

- Determining the class of a reflection: (clipper::HKL\_class)
  - Loop over all (primitive) symmetry operators.
  - If  $\mathbf{S}_k^T \underline{h} = -\underline{h}$ , the reflection is centric.
    - Calculate its allowed phases.
  - If  $\mathbf{S}_k^T \underline{h} = \underline{h}$ :
    - If  $\underline{h}^T \underline{S}_k$  is an integer, increase the enhancement by 1, otherwise the reflection is a systematic absence.
  - Enhancement is increased by the number of centerings.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry: $hkl$

- Transforming a reflection about reciprocal space:  

$$|E(\mathbf{S}_k^T \underline{h})| = |E(\underline{h})|$$

$$\phi(\mathbf{S}_k^T \underline{h}) = \phi(\underline{h}) - 2\pi \underline{h}^T \underline{S}_k$$
- From these, calculate the transformations for other types of data, e.g. A,B, Hendrickson-Lattmann coefficients.
  - Clipper: when a reflection data type is defined, its behavior under phase shift or Friedel inversion is also defined. With these the reflection can be transformed about reciprocal space by 'magic'.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry: $hkl$

Reciprocal space asymmetric units (ASUs).

- P1:
  - For the most common calculations, we don't need to store both a reflection and its Friedel opposite (since  $E(-\underline{h}) = E(\underline{h})^*$ ). Even for anomalous data, we usually store  $E(\underline{h})$  and  $E(-\underline{h})$  together.
  - Therefore, we only need to store a hemisphere of data.



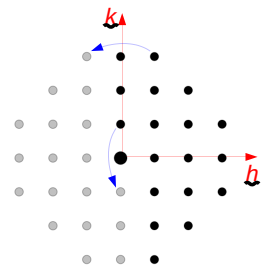
Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry: $hkl$

Reciprocal space asymmetric units (ASUs).

- P1: Hemisphere of data.
  - But even that isn't simple.
  - Use for example:
    - ( $I > 0$ ) or
    - ( $I = 0$  and  $h > 0$ ) or
    - ( $I = 0$  and  $h = 0$  and  $k > 0$ )



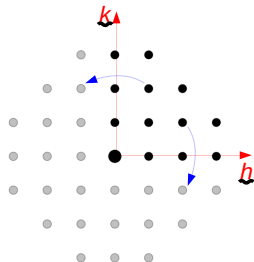
Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry: $hkl$

Reciprocal space asymmetric units (ASUs).

- **P 2 2**: (P 2 2 2)
  - $h \geq 0$  and  $k \geq 0$  and  $l \geq 0$
- **P 3**: (P 3)
  - $(h \geq 0 \text{ and } k \geq 0)$  or
  - $(h = 0 \text{ and } k = 0 \text{ and } l \geq 0)$
- **P 4 3 2**: (P 4 3 2)
  - $k \geq l$  and  $l \geq h$  and  $h \geq 0$



Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry: $hkl$

Implementation:

- Calculate point-group, and 'change-of-basis' to get to standard setting, then use ASU for that point-group on transformed  $hkl$ . (13 tabulated ASUs).
  - **CCTBX**, **CCP4**
- Calculate oriented point-group, and then use ASU for that point-group  $hkl$ . (51 tabulated ASUs).
  - **Clipper** optimization.
  - Sanity check by using a sample set of reflections.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry:

General implementation points:

- **CCTBX** is fully space-group general, and will handle made-up settings without difficulty.
- **Clipper** is fully space-group general, and will handle made-up settings without difficulty (except it won't name a space-group from the symops unless they match one from a list of common and uncommon settings).
- **CCP4** uses a data file of common and uncommon settings, and won't handle anything else.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry:

Implementation: Maps

- Typical map access modes...
  - Loop over each unique grid point in turn.
  - Access a grid point, and then one of its neighbors.
  - Access grid points at random.
- Map data objects are ideally specialized to the access pattern required.
  - But useful compromises are possible.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

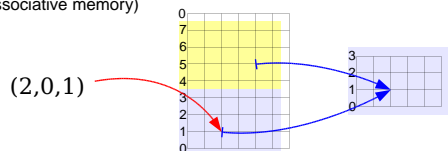
Sienna/Crystal Symmetry

## Crystallographic symmetry:

Implementation: Maps

- e.g. for random access, use mod() to get grid coordinates into the unit box, and then use a unit cell grid of pointer pointing to the density value in an asymmetric unit list.
  - Cost: an extra memory access.
  - To save memory share lookup tables between maps using the same grid and space-group.

(Associative memory)



Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry:

Implementation: Maps

- **Clipper approach**: random access is rare. Store an ASU grid with pointers round the edge so that when we run out of the grid we know where to get the next density.
  - No additional lookup.
  - Good for sequential access and neighbor access.
  - Lower overhead.
  - For full-cell random access, a search over symops is required (or expand to P1).

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Crystal Symmetry

## Crystallographic symmetry:

### Implementation: Reflections

- Store a list of reflections with h,k,l?
  - Good for sequential access.
  - Efficient storage of ASU.
- Or store a 3D array of reflections?
  - Good for random access.
  - Inefficient storage of ASU.
  - Sequential access may be harder.

## Crystallographic symmetry:

### Implementation: Reflections

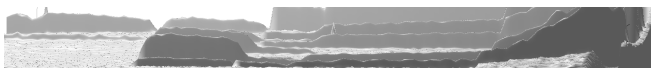
- Clipper approach: Store a list of h,k,l and reflection data values of arbitrary types. But also provide a ragged 3D index array (ASU only) for when random access is required.
  - Fairly compact.
  - For full-sphere random access, a search over symops is required.

## Crystallographic symmetry:

### Summary:

- Symmetry is fundamental to crystallographic calculation in both real and reciprocal space.
- We use space-group symbols, calculations use the symmetry operators.
- Symmetry involves a lot of book-keeping, which may be avoided by good enough class design.
  - General designs are useful for many/most purposes.
  - However, optimal solutions may be problem specific.





## Profile refinement Least-squares analysis and beyond



Bill David, ISIS Facility,  
Rutherford Appleton Laboratory, UK

IUCr Computing School Siena 18-23 August 2005



## Part I

### When and why do we use least-squares analysis in crystallography?

#### Using least-squares analysis: a basic part of crystallography

- It's worked all right up till now!
  - How accurate are your structural parameters?
- The fit looks pretty good!
  - How good are your data really?
  - How good, how complete is your model?
- If it ain't broke don't fix it!

Bert Lance 1977

"Learn the fundamentals of the game and stick to them. Band-Aid remedies never last."

Jack Nicklaus

- fundamental parameters
  - Pearson VIIs are good but fundamental parameters are better
  - "One of the intrinsic benefits of the fundamental parameters approach is that it is easily adapted to any laboratory diffractometer. Good fits can normally be obtained over the whole  $2\theta$  range without refinement using the known properties of the diffractometer (i.e. slit sizes, diffractometer radius and so on) and the emission profile."
  - you can understand and explain the peak shape function
- fundamental statistics
  - optimisation by plausible reasoning
  - probabilistic reasoning for least-squares analysis and beyond
  - minimise empiricism - no *deus ex machina*

#### What is least squares analysis?

$$\chi^2 = \sum_i \left( \frac{\text{obs}(i) - \text{calc}(i)}{\text{esd}(i)} \right)^2$$

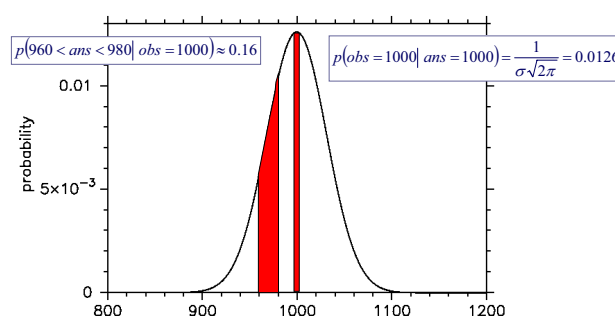
When do we use least squares analysis?

extremely specific

extremely broad

Least squares analysis has its roots in the assumption that the errors in the data follow a Gaussian (normal) probability distribution function.

#### Least-squares analysis equates to the data following a Gaussian probability distribution



Finding the most probable solution = maximising the probability

likelihood 
$$p(D_i | \mu, \sigma = \sqrt{\mu}) = \prod_{i=1}^N \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma_i^2} (D_i - \mu)^2\right)$$

Maximising the likelihood = minimising  $-(\log\text{-likelihood})$

Minimise 
$$-\ln(p(D_i | \mu, \sigma = \sqrt{\mu})) = \sum_{i=1}^N \left( \frac{1}{2} \ln(2\pi) + \ln \sigma_i + \frac{1}{2\sigma_i^2} (D_i - \mu)^2 \right) \\ \cong C + \sum_{i=1}^N \left( \frac{1}{2\sigma_i^2} (D_i - \mu)^2 \right)$$

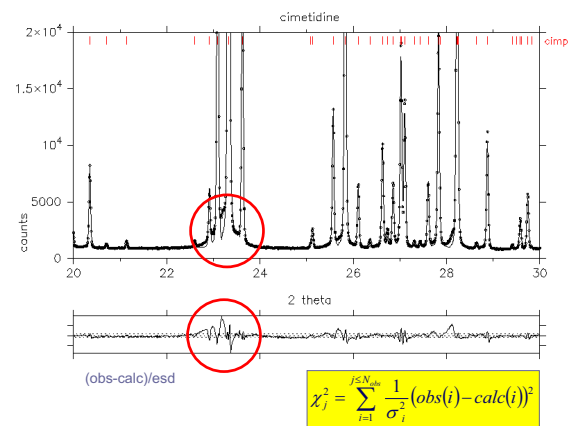
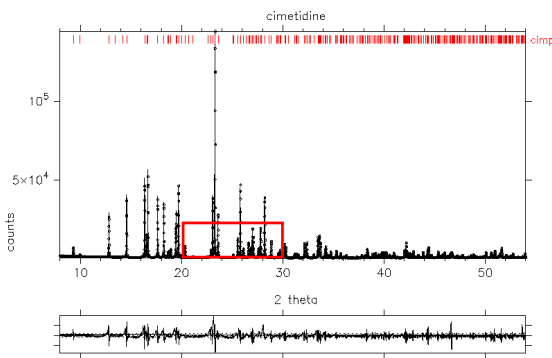
c.f. least squares minimisation

Minimise 
$$\chi^2 = \sum_{i=1}^N \left( \frac{1}{\sigma_i^2} (D_i - \mu)^2 \right)$$

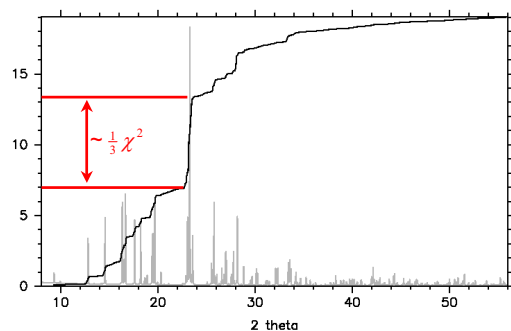
## Part II

What if the fit isn't that good?

What if the fit isn't that good?

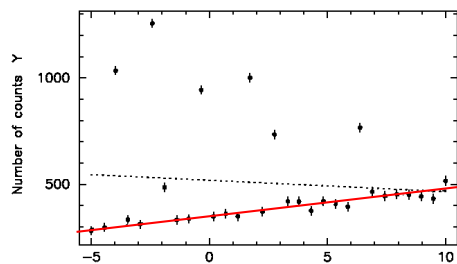


$$\chi_j^2 = \sum_{i=1}^{j \leq N_{obs}} \frac{1}{\sigma_i^2} (obs(i) - calc(i))^2$$



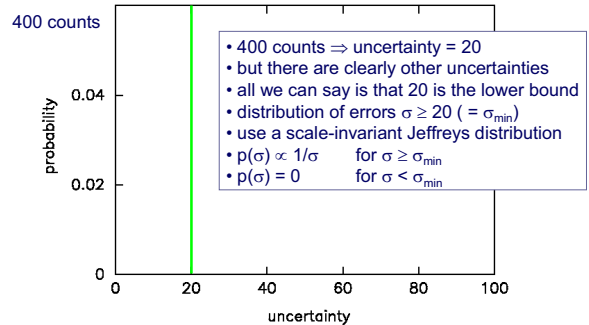
What if the fit isn't that good

- Is it wise to have 150,000 counts in the biggest peak and 5000 counts in a very highly structured background?
  - No! Redo the experiment!
- Collect all Bragg peaks with similar fractional accuracy
  - variable counting time to give  $E/\sigma(E)$  constant
- If accuracy and precision are required be prepared to
  - comprehensively model structure and microstructure
  - perform fundamental line-shape analysis
  - undertake detailed “fundamental” background analysis
- If all else fails - use statistics / plausible reasoning!

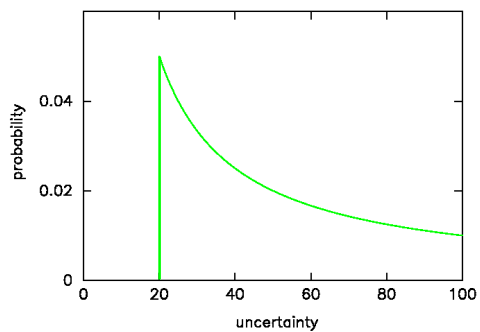


## What's gone wrong?

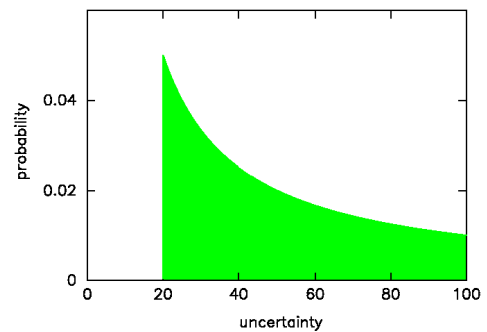
- We've performed a least-squares analysis and implicitly assumed that all errors follow a Gaussian PDF
- We've been certain about our uncertainties!



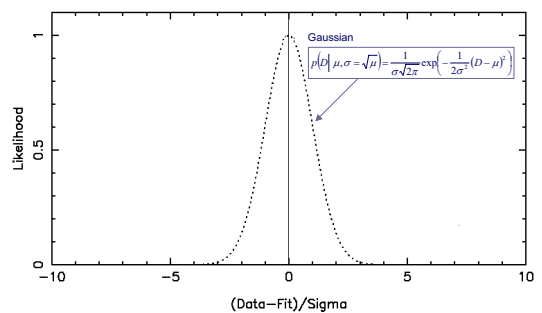
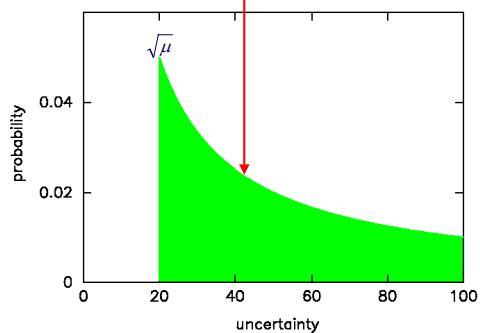
- Jeffreys prior
- $p(\sigma) \propto 1/\sigma$  for  $\sigma \geq \sigma_{\min}$
- $p(\sigma) = 0$  for  $\sigma < \sigma_{\min}$

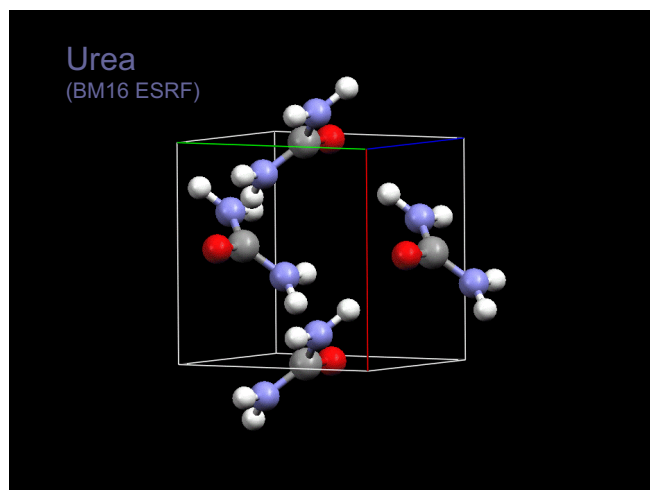
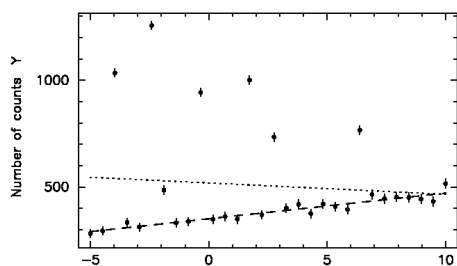
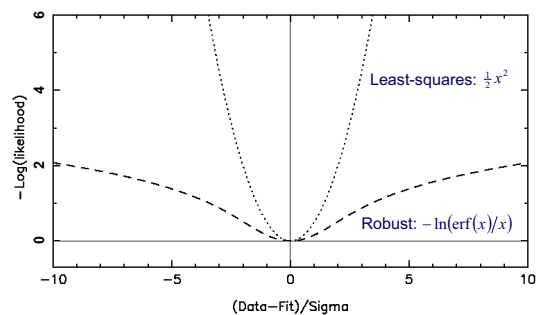
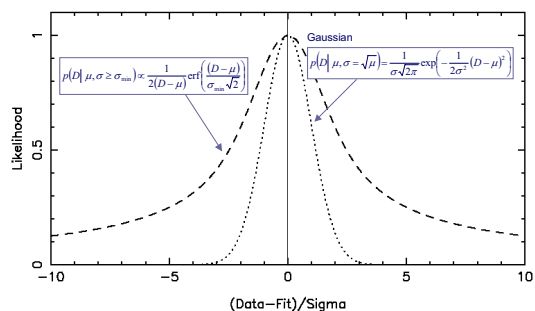


- Jeffreys prior
- $p(\sigma) \propto 1/\sigma$  for  $\sigma \geq \sigma_{\min}$
- $p(\sigma) = 0$  for  $\sigma < \sigma_{\min}$

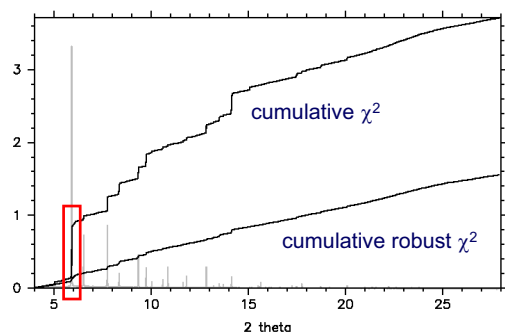


$$p(D|\mu, \sigma \geq \sqrt{\mu}) = \int_{\sigma_{\min}=\sqrt{\mu}}^{\infty} \boxed{\text{prob}(\sigma)} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(D-\mu)^2\right) d\sigma$$





Urea (BM16 ESRF)



Urea (BM16 ESRF)

	SXXD	Least Squares	LS-SXXD	Robust	R-SXXD
C1 z	0.3328(3)	0.3236(9)	-0.0092(10)	0.3319(13)	-0.0009(14)
O1 z	0.5976(4)	0.6013(5)	0.0037(6)	0.5984(7)	0.0008(8)
N1 x	0.1418(2)	0.1405(3)	-0.0013(4)	0.1423(7)	0.0005(7)
z	0.1830(2)	0.1807(5)	-0.0023(6)	0.1813(7)	-0.0017(7)
C1 U11	0.0353(6)	0.0348(20)	-0.0005(20)	0.0329(40)	0.0024(40)
U33	0.0155(5)	0.0396(30)	0.0241(30)	0.0413(40)	0.0258(40)
U12	0.0006(9)	0.0205(30)	0.0199(30)	0.0128(40)	0.0122(40)
O1 U11	0.0506(9)	0.0749(16)	0.0243(18)	0.0617(30)	0.0111(30)
U33	0.0160(6)	0.0080(14)	-0.0080(15)	0.0090(20)	-0.0070(20)
U12	0.0038(18)	0.0052(20)	0.0014(30)	-0.0011(35)	-0.0049(35)
N1 U11	0.0692(6)	0.0627(15)	-0.0068(18)	0.0697(25)	0.0005(25)
U33	0.0251(4)	0.0460(22)	0.0211(22)	0.0365(30)	0.0114(30)
U12	-0.0353(7)	-0.0252(18)	0.0101(20)	-0.0361(30)	-0.0008(30)
U13	-0.0003(3)	-0.0015(11)	-0.0012(12)	-0.0029(15)	-0.0026(15)

  = diff > 4σ

9/14 > 4σ

1/14 > 4σ

## Part III

### What if the model is incomplete?

- unknown impurity phase
- unknown fragment in crystal structure
  - e.g. disordered guest in zeolite
  - unknown waters of hydration
  - disordered oxygen in high  $T_c$  material
  - incomplete models in structure solution
  - $F_{\text{calc}}$  has uncertainty as well as  $F_{\text{obs}}$
  - (c.f. maximum likelihood in structural biology)

### Gaussian probability distribution

We're happy that the data errors follow a Gaussian distribution

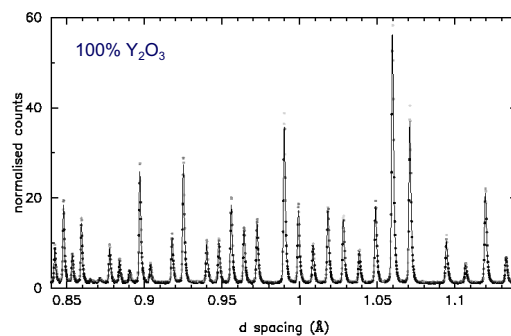
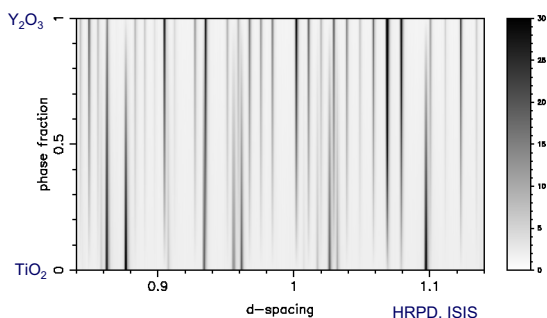
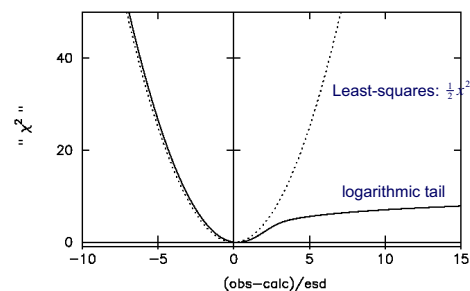
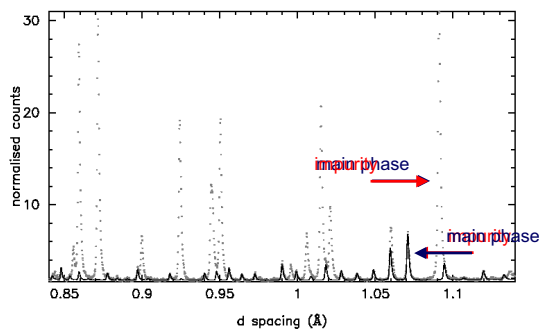
$$p(D|M, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(D-M)^2\right)$$

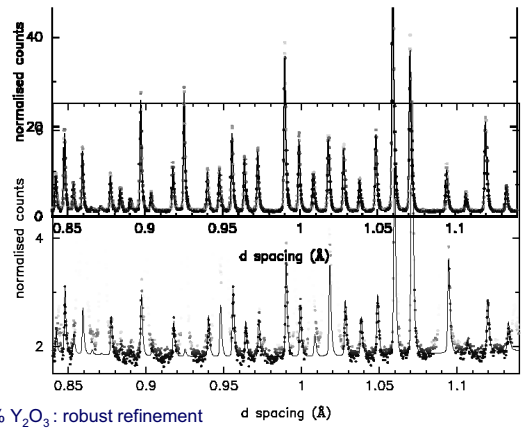
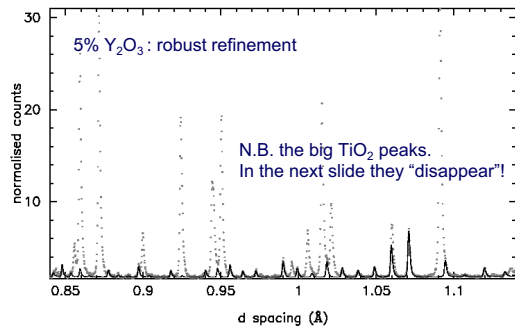
Our problem is that the model is incomplete. We have a known phase with contribution,  $P$ , and an unknown (positive) component,  $A$ .  
i.e.  $M=P+A$

$$p(D_i|P, \sigma) = \int \text{prob}(A) \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(D-(P+A))^2\right)$$

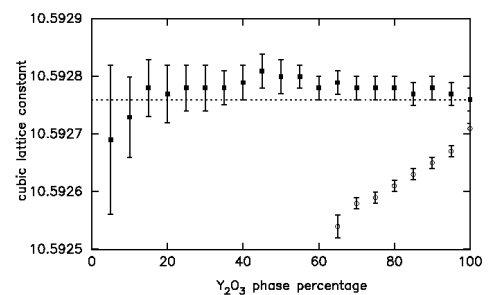
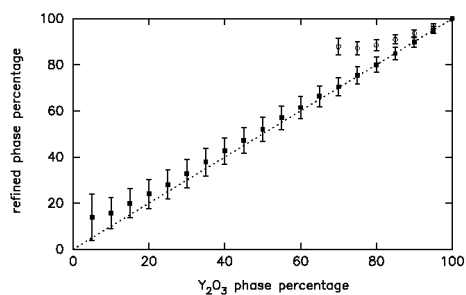
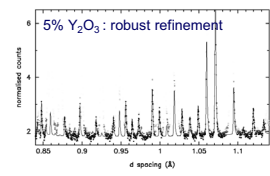
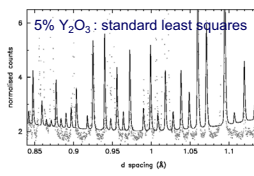
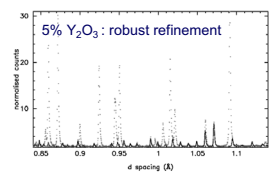
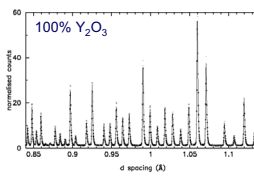
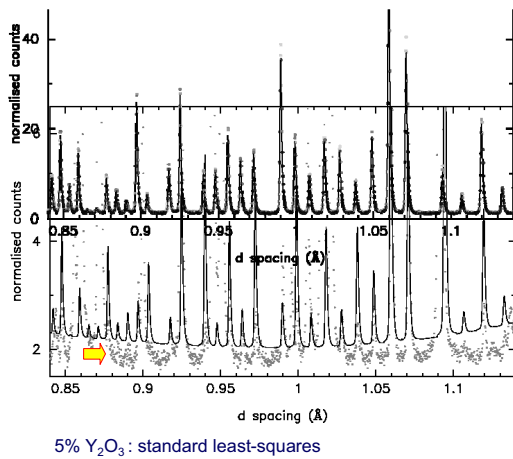
Use a scale invariant Jeffreys  $1/A$  prior for  $\text{prob}(A)$

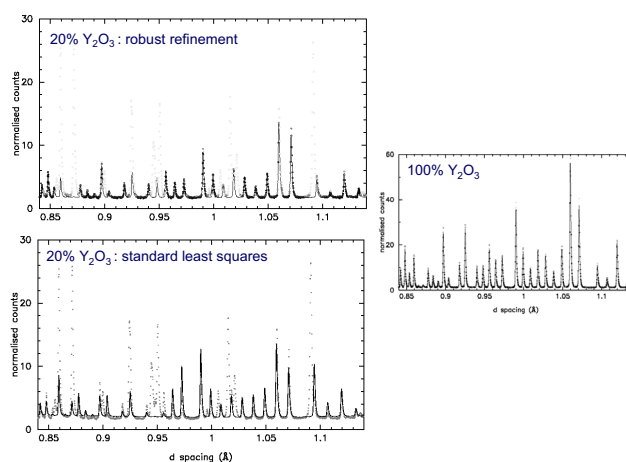
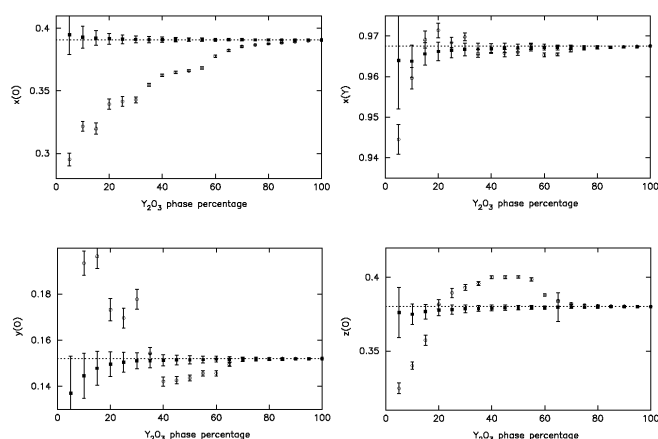
### What if there's an impurity phase?





The darkness of the dots indicates their relative impact in the robust analysis. Large positive (obs-calc) points (i.e. mostly impurity peaks) are “invisible”.





## Summary of Part III

If the model is incomplete, careful construction of an appropriate probability distribution function can bring significant improvements over standard least-squares analysis.

*J. Appl. Cryst.* (2001). 34, 691-698

### Robust Rietveld refinement in the presence of impurity phases

W. I. F. David

**Abstract:** A modified least-squares analysis is presented that allows reliable structural parameters to be extracted from a powder diffraction pattern even in the presence of a substantial unmodelled impurity contribution. The algorithm is developed within the context of Bayesian probability theory. Experimental points that fall above those calculated, and are thus more probably from impurity peaks, are systematically down-weighted. This approach is illustrated with a two-phase example.

*Acta Cryst.* (2002). A58, 316-326

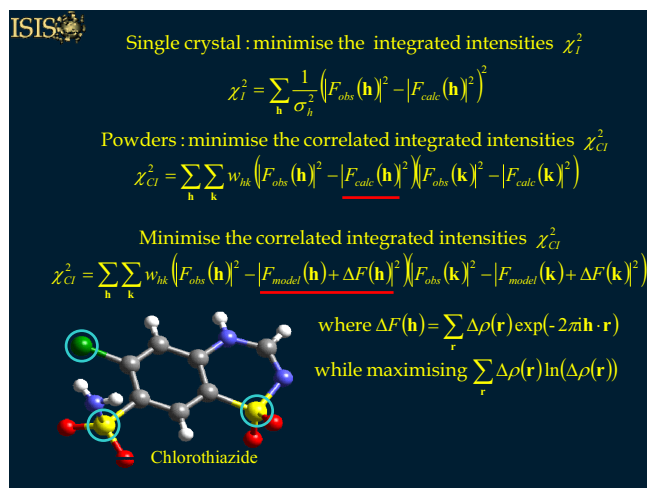
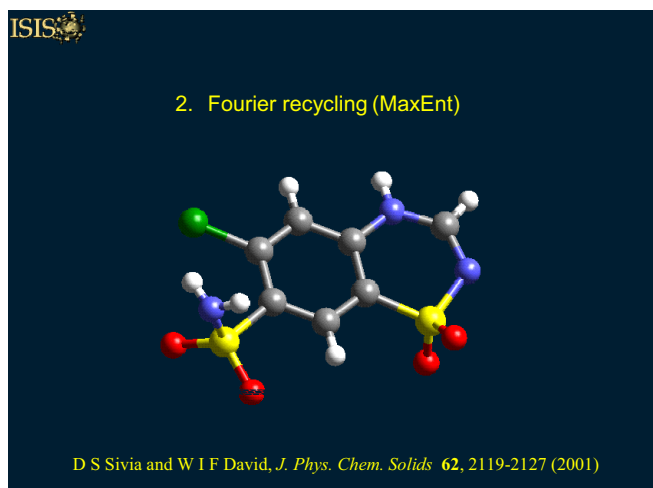
### A maximum-likelihood method for global-optimization-based structure determination from powder diffraction data

A. J. Markvardsen, W. I. F. David and K. Shankland

**Abstract:** A maximum-likelihood algorithm has been incorporated into a crystal structure determination from a powder diffraction data framework that uses an integrated-intensity-based global optimization technique. The algorithm is appropriate when the structural model being optimized is not a complete description of the crystal structure under study.

## Conclusions

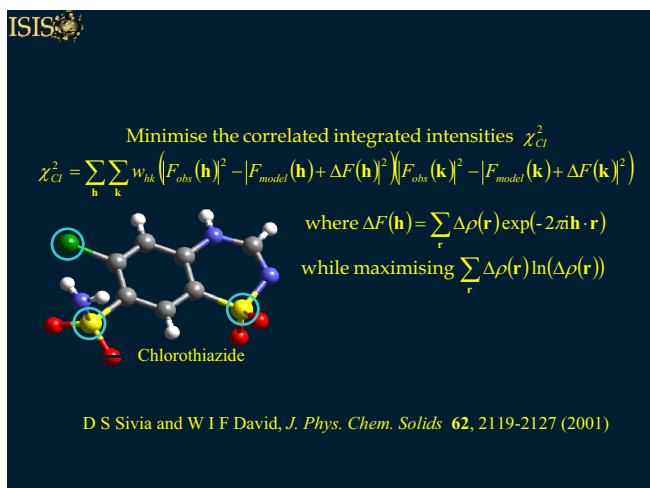
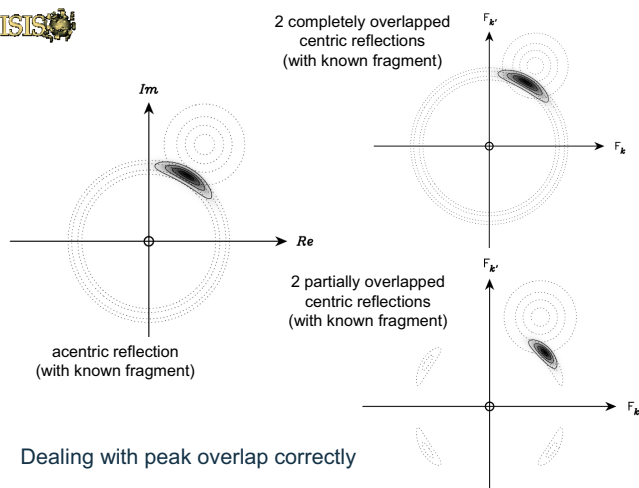
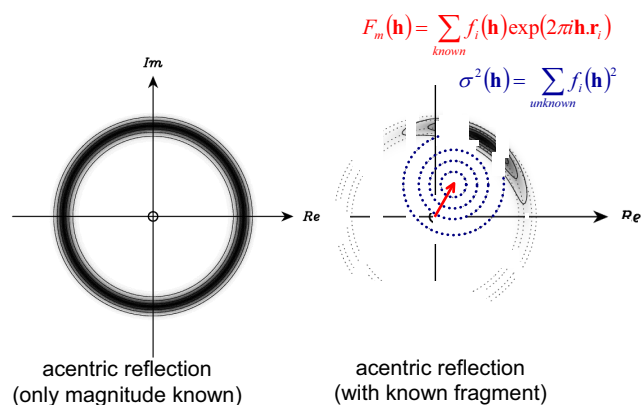
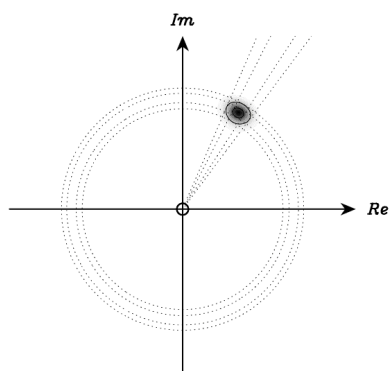
- least-squares is fairly ubiquitous but not always the most appropriate minimisation metric.
- try to keep things simple
  - do the best experiment possible
  - develop as complete a model as possible
- be as certain as possible about your uncertainties (probability distribution functions)
- be prepared to go beyond least-squares!



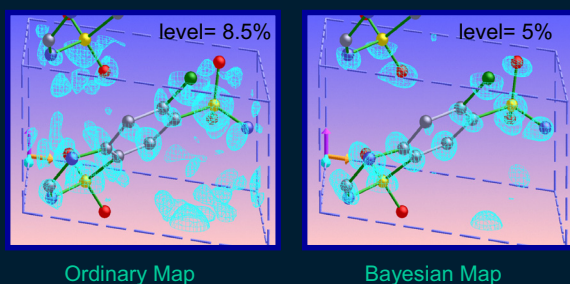




## The importance of phase information

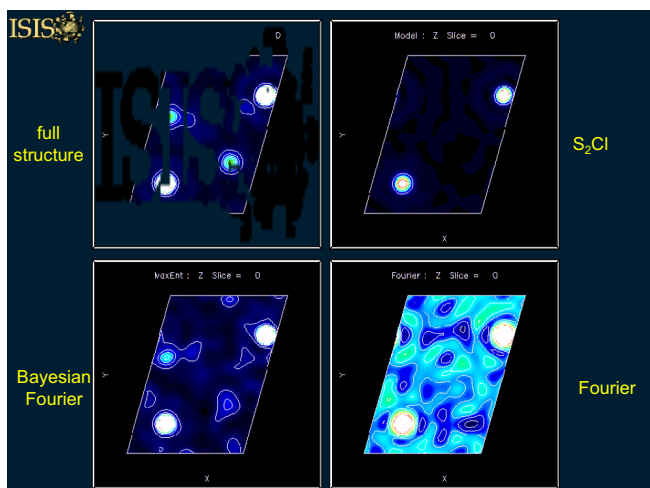


## Bayesian map reconstruction from powder diffraction data

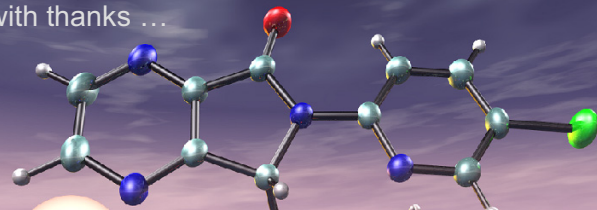


Chlorothiazide:  $\text{C}_7\text{H}_5\text{N}_3\text{O}_4\text{S}_2\text{Cl}$

Bayesian approach:  $\text{S}_2\text{Cl}$  (32% scattering) to full structure



with thanks ...



M Brunelli, A N Fitch, J P Wright (ESRF)

A A Coelho

N Shankland, A Kennedy (Strathclyde)

C Pulham (Edinburgh)

K Shankland, A J Markvardsen, D S Sivia (ISIS)

Zopiclone

D S Sivia, Data Analysis: A Bayesian Tutorial, OUP, ISBN 0-19-851889-7

# Fourier Transforms in Crystallography

They are EVERYWHERE

L. F. Ten Eyck

2005 IUCr Crystallographic Computing School

FFTs

## Outline

- 1 Introduction
  - Physics
  - Mathematics

Navigation icons

FFTs

## Outline

- 1 Introduction
  - Physics
  - Mathematics

Navigation icons

FFTs

## Diffraction is a Physical Fourier Transform

- Scattering of a wave by an object *is* the Fourier transform of the scattering function.

$$F(h) = \int_{-\infty}^{+\infty} f(x) \exp(2\pi i h \cdot x) dx$$

- Fourier transform of periodic functions are discrete.

$$F(h) = 0 \text{ if } h \text{ is not an integer.}$$

- Crystals are (**MOSTLY**) periodic.
- The scattering function can thus be reconstructed by summing the Fourier series

$$f(x) = \sum_{h=-\infty}^{\infty} F(h) \exp(-2\pi i h \cdot x)$$

Navigation icons

FFTs

## Outline

- 1 Introduction
  - Physics
  - Mathematics

FFTs

## General Properties of Fourier Transforms

The FT of a real function is Hermitian Symmetric, and Vice-Versa

- If  $f(x)$  is real,  $F(h) = F^*(-h)$ .
- If  $F(h) = F^*(-h)$ ,  $f(x)$  is real.

### Gaussian Functions

- If  $f(x) = A \exp(-ax^2)$  then

$$F(h) = \frac{A}{\sigma^3} \exp(-\pi h^2 / \sigma^2)$$

where  $\sigma^2 = a/4\pi$ .

- Wide Gaussians in real space are narrow in reciprocal space
- Narrow Gaussians in real space are wide in reciprocal space

Navigation icons

FFTs

## General Properties of Fourier Transforms

Continued...

## Convolution Theorem

- The Fourier transform of a convolution is the product of the Fourier transforms of the functions

$$C(\tau) = \int_{-\infty}^{\infty} f(t)g(\tau - t)dt$$

$$C(h) = F(h)G(h)$$

$$\therefore C(\tau) = \int_{-\infty}^{+\infty} F(h)G(h) \exp(2\pi i h \cdot \tau) dh$$

FFTs

## Convolutions

The Second Most Important Concept in Crystallography

## Crystals as Convolutions

- Molecules: Atomic distribution functions and point locations
- Unit cells: Mosaicity distribution
- Material heterogeneity: Distribution of atomic types
- Crystal shape

## Measurements as Convolutions

- Beam Profile
- Aperture and Collimator
- Dispersion

FFTs

## Finite Discrete Fourier Transforms

## Definition

Let  $X(t)$  be a sequence of  $N$  points with  $t = 0, 1, \dots, N-1$ . The Discrete Fourier Transform of  $X(t)$  and the inverse DFT are

$$X(\hat{t}) = \sum_{t=0}^{N-1} X(t) \exp(2\pi i t \hat{t} / N)$$

$$X(t) = \frac{1}{N} \sum_{\hat{t}=0}^{N-1} X(\hat{t}) \exp(-2\pi i t \hat{t} / N)$$

The function  $e(x) = \exp(2\pi i x)$  has the properties that  $e(x) = 1$  if  $x$  is an integer,  $e(-x) = e^*(x)$ ,  $e(x+y) = e(x)e(y)$ , and thus  $e(x+N) = e(N)e(x) = e(x)$ .

FFTs

## Properties of DFT's

DFT's are DIFFERENT

## Continuous

Orthogonality:

$$\int_{-\infty}^{+\infty} e(t\hat{t})e(-tu)dt = \delta(\hat{t}-u)$$

Periodicity:

$$\text{If } f(x) = f(x + Na)$$

 $F(h)$  is NOT periodic

## Discrete

Orthogonality:

$$\sum_{t=0}^{N-1} e(t\hat{t})e(-tu) = N\delta(\text{mod}(\hat{t}-u, N))$$

Periodicity: for all integer  $k$ ,

$$X(t) = X(t + kN)$$

$$X(\hat{t}) = X(\hat{t} + kN)$$

FFTs

## Factoring DFT's

Let  $N=AB$ . Then  $\hat{t} = \hat{a} + \hat{b}A$  and  $t = b + aB$  cover all values of  $\hat{t}$  and  $t$  as  $\hat{a}, \hat{b} = 0, \dots, A-1$  and  $b, a = 0, \dots, B-1$ . Then

$$\begin{aligned} \hat{X}(\hat{a} + \hat{b}A) &= \sum_{b=0}^{B-1} \sum_{a=0}^{A-1} X(b + aB) e\left(\frac{(b + aB)(\hat{a} + \hat{b}A)}{AB}\right) \\ &= \sum_{b=0}^{B-1} \sum_{a=0}^{A-1} X(b + aB) e\left(\frac{\hat{a}b}{AB} + \frac{\hat{a}a}{A} + \frac{\hat{b}b}{B} + a\hat{b}\right) \\ &= \sum_{b=0}^{B-1} \sum_{a=0}^{A-1} X(b + aB) e\left(\frac{\hat{a}b}{AB}\right) e\left(\frac{\hat{a}a}{A}\right) e\left(\frac{\hat{b}b}{B}\right) \end{aligned}$$

because  $e(a\hat{b}) = 1$ .

Page 157

FFTs

## Properties of DFT's

## Factoring (cont.)

$$\hat{X}(\hat{a} + \hat{b}A) = \sum_{b=0}^{B-1} e\left(\frac{\hat{b}b}{B}\right) \left\{ e\left(\frac{\hat{a}b}{AB}\right) \sum_{a=0}^{A-1} X(aB + b) e\left(\frac{\hat{a}a}{A}\right) \right\}$$

This reduces the  $N^2$  problem of computing  $N$  Fourier coefficients, each requiring a sum of  $N$  terms, to  $B$  problems of size  $A^2$  followed by  $A$  problems of size  $B^2$ .

$$BA^2 + AB^2 = AB(A + B) \approx N \log_2 N$$

FFTs

## Properties of DFT's

Factoring can be used to design code that

- Is specialized for real and Hermitian symmetric data
- Incorporates crystallographic symmetry elements
- Factors of two and four do not require multiplication
- Details in Ten Eyck (1973), *Acta Cryst.* **A29**, 183-191.

Navigation icons: back, forward, search, etc.

FFTs

## Properties of DFT's

## DFT's are Convolutions

- A sequence of samples is a convolution of the sampled function with a  $\delta$  function. If  $f(u)$  is periodic with period  $a$ ,

$$X(t) = f\left(\frac{ta}{N}\right) = \int_0^a f(u) \delta\left(t - u \frac{N}{a}\right) du$$

- $\hat{X}(\hat{t})$  is the sum of Fourier coefficients spaced  $N$  points apart.

$$\hat{X}(\hat{t}) = \sum_{k=-\infty}^{+\infty} F(\hat{t} + kN)$$

- This places a lower limit on useful values of  $N$ .

Navigation icons: back, forward, search, etc.

FFTs

## Properties of DFT's

## Multiple conventions are in use

- Normalization:
  - None
  - $1/\sqrt{N}$  in each direction
  - $1/N$  in one direction, 1 in the other
- Sign of exponent
  - $\exp\left(+2\pi i \frac{ft}{N}\right)$  forward,  $\exp\left(-2\pi i \frac{ft}{N}\right)$  reverse
  - $\exp\left(-2\pi i \frac{ft}{N}\right)$  forward,  $\exp\left(+2\pi i \frac{ft}{N}\right)$  reverse

Navigation icons: back, forward, search, etc.

FFTs

## Calculation of Fourier Summations

## Summations

The FFT is a very fast way of calculating Fourier summations.

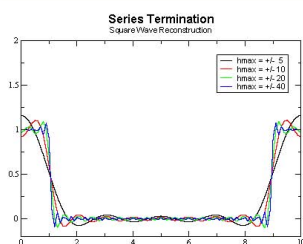
- Direct cost is  $MN$  where  $M$  is the number of Fourier coefficients and  $N$  is the number of grid points.
- Typical values are  $M = 10,000$  and  $N = 128^3 = 2097152$ .
- FFT cost is  $N \log_2 N$ .
- Cost ratio for this case is  $\frac{M}{\log_2 N} = 476$  in favor of FFT.

Navigation icons: back, forward, search, etc.

FFTs

## Series Termination

## Summations



- Square wave of length 100
- Fourier series truncated at 5, 10, 20, and 40
- Note Gibbs' Phenomenon – the overshoot is **constant**

Navigation icons: back, forward, search, etc.

FFTs

## Calculation of Density Maps

## Tradeoffs

## Memory Requirements

- Can calculate only an asymmetric unit
  - But sometimes P1 is convenient
- Transform in place
- Hermitian to Real saves factor of 2

## Speed

- Hermitian to Real saves factor of 2
- Use of symmetry saves factor of  $N_{\text{symp}}$

Navigation icons: back, forward, search, etc.

FFTs



## Structure Factors

## Procedure

- 1 Determine grid spacing and "blur" from resolution limits
- 2 Generate model density from coordinates
  - 1 Use multiple Gaussian scattering factors
  - 2 Transform of Gaussian is Gaussian
  - 3 Mapping of elliptical (anisotropic) density is no more work than spherical density
  - 4 Wrap across boundaries of asymmetric unit and unit cell
- 3 Calculate Real to Hermitian Fourier transform
- 4 Remove "blur"

FFTs

## Structure Factors

## Resolution and "Blur"

## Resolution and Accuracy

- 1 Aliasing causes serious contamination by high-order harmonics

$$\mathbf{F}_{FFT}(h) = \sum_{t=-\infty}^{+\infty} \mathbf{F}_c(h + tN)$$

- 2 Structure factors have generally Gaussian distribution – as in Wilson plots.
- 3 Convolution with a Gaussian "blur" causes the error to fall off faster than linearly.

FFTs

## Structure Factors

## Resolution and "blur"

- 1 Add "blur"  $B^0$  to every atom, noting that  $(\sin \theta)/\lambda = h/2a$

$$\begin{aligned} \mathbf{F}_B(h) &= \exp(-B^0 h^2/4a^2) \mathbf{F}(h) \\ \mathbf{F}_B(h+N) &= \exp(-B^0(h+N)^2/4a^2) \mathbf{F}(h+N) \\ \mathbf{F}_B(h-N) &= \exp(-B^0(h-N)^2/4a^2) \mathbf{F}(h-N) \end{aligned}$$

- 2 The fractional contamination by the two largest contaminating structure factors can be made as small as desired by appropriate choice of  $B^0$  and  $N$ .
- 3 Choice depends on resolution and cell edge.
  - 1  $N \geq 3(a/h_{max})$  and  $B^0 > 15$  is **minimum** for medium resolution
  - 2 At high resolution,  $B^0 \approx 5$  and  $N$  large is required (Tronrud).
- 4 Details in Ten Eyck, *Acta Cryst.* **A33**, 486-492 (1977)

FFTs

## Least Squares Gradients

- 1 If  $\Phi(\mathbf{p}) = \sum_h W(h) [k |\mathbf{F}_o(h)| - |\mathbf{F}_c(h, \mathbf{p})|]^2$ , the derivatives with respect to the parameters  $\mathbf{p}$  are given by

$$\frac{d\Phi(\mathbf{p})}{d\mathbf{p}} = -2 \sum_h W(h) [k |\mathbf{F}_o(h)| - |\mathbf{F}_c(h, \mathbf{p})|] \frac{d|\mathbf{F}_c(h, \mathbf{p})|}{d\mathbf{p}}$$

- 2  $\mathbf{F}_c(h, \mathbf{p}) = \sum_{atoms} f_k(h) \exp(-B_k \sin^2 \theta / \lambda^2) \sum_{pos} \exp(2\pi i \mathbf{h}^T \mathbf{R}_j \mathbf{x}_k)$  so the product above is the product of two Fourier transforms – for the difference map, and for the density due to each atom.
- 3 The sum of products of Fourier transforms can be calculated by convoluting the gradients of the model atomic density with the difference map.

Details are in Tronrud, Ten Eyck and Matthews, *Acta Cryst.* **A43**, 489-501 (1987).

FFTs

## Random Observations

## FFTW

- 1 FFTW (<http://www.fftw.org>) is OUTSTANDING.
- 2 FFTW IS NOT DETERMINISTIC if it is tuning.

## Prototype and Explore

MATLAB (\$\$\$) and Octave (free) both have excellent facilities for playing with Fourier transforms

## Links

- 1 MathWorld: <http://mathworld.wolfram.com> – lots of math; use the search functions.
- 2 Wikipedia: <http://www.wikipedia.org> – excellent material on DFT's and pointers to more.
- 3 FFTW: again, excellent discussion.

FFTs

## Maximum Likelihood

### Maximum Likelihood in X-ray Crystallography

Kevin Cowtan  
[cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Maximum Likelihood

Inspired by Airlie McCoy's lectures.  
<http://www-structmed.cimr.cam.ac.uk/phaser/publications.html>

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Maximum Likelihood

- Science involves the creation of hypothesis (or theories), and the testing of those theories by comparing their predictions with experimental observations.
- In many cases, the conclusions of an experiment are obvious – the theory is supported or disproven.
- In other cases, results are much more marginal.  
e.g. How big a sample size do we need to distinguish a successful drug from placebo effect?

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Maximum Likelihood

- In order to correctly understand the impact of our experiments on our theories, we need some knowledge of statistics.
- This is especially necessary in crystallography, since we have:
  - a very weak signal:  
(the observed magnitudes)
  - a great deal of noise:  
(the missing phases + measurement errors)
  - from which we are trying to test a very detailed hypothesis:  
(the position of every atom in the unit cell)

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Maximum Likelihood

- Given the uncertainties of the data, we cannot usually determine whether a hypothesis is right or wrong – only how likely it is to be right:  
(The probability of the hypothesis.)
- In order to do this, our hypothesis must be detailed enough for us to work out how likely we would have been to get the results we observe, assuming that the hypothesis is true.
- We then use Bayes' theorem to determine the probability.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Maximum Likelihood

- Examples:
  - Hypothesis: The observed X-ray data arises from a molecule which is roughly homologous to this known molecule in this orientation in the cell.  
(Molecular replacement – how probable is a given model)
  - Hypothesis: The position of this atom (and its neighbors better explains the X-ray data when moved in this direction.  
(Refinement – what is the relative probability of two very similar models. Includes heavy-atom refinement.)
- In fact all problems come down to the comparison of different hypotheses.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

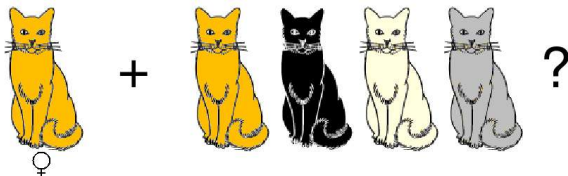
Sienna/Maximum Likelihood



# Understanding Likelihood

Example:

- We have a cat. She has a kitten. We don't know who the father is, but there are four possibilities in the neighborhood.
- What can we say about the color of the father from the color of the kitten?



Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Maximum Likelihood

# Understanding Likelihood

Cat genetics is complex: we will simplify.

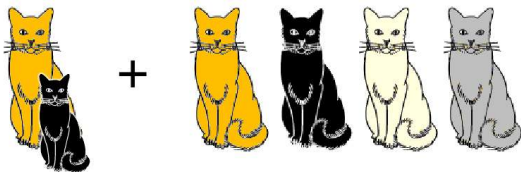


Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Maximum Likelihood

# Understanding Likelihood

One black kitten: which is the father?



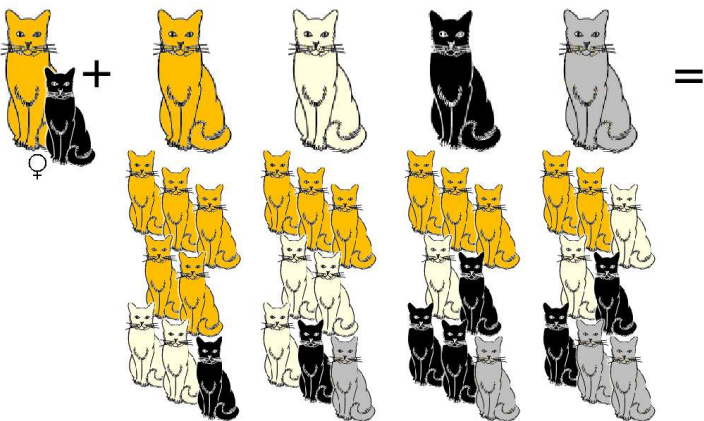
Actually, it could be any one, since they may all carry the appropriate genes. But they are not all equally probable.

We need some more information.

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Maximum Likelihood

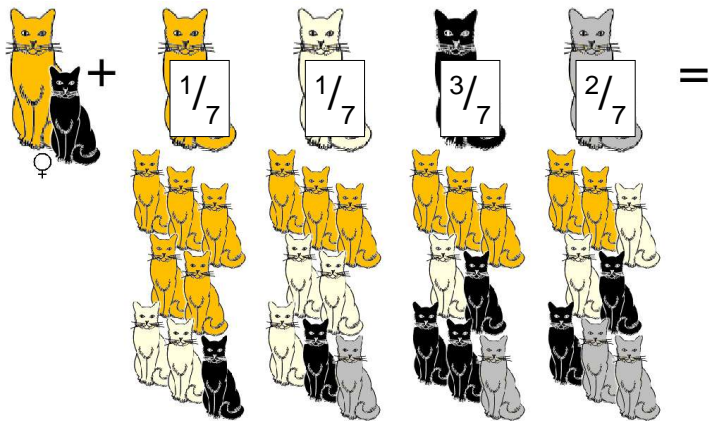
# Understanding Likelihood



Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Maximum Likelihood

# Understanding Likelihood



Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Maximum Likelihood

# Understanding Likelihood

An extremely clever transformation has occurred:

- I gave you the probability of a kitten being a particular color, given that we know the colors of the father.  
 $P(\text{kitten color} \mid \text{father color})$
- You gave me the probability of the father being a particular color, given that we know the color of the kitten.  
 $P(\text{father color} \mid \text{kitten color})$
- $P(x \mid y)$ : The probability of  $x$ , given  $y$ .

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Maximum Likelihood

## Understanding Likelihood

### This is a simple experiment:

- The result of the experiment is our observed data: the color of the kitten.
- The hypothesis is concerning the color of the cat. We make 4 hypotheses about the father (orange, black, cream, grey) and calculate the probability of each.
- We can work out  $P(\text{data} | \text{hypothesis})$
- We want to know  $P(\text{hypothesis} | \text{data})$

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Understanding Likelihood

How do we do this sort of maths for a general problem?

- Use Bayes' theorem:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

- Proof: The probability of  $x$  and  $y$  is the probability of  $x$  given  $y$  multiplied by the probability of  $y$ .

$$P(x, y) = P(x|y)P(y)$$

$$P(y, x) = P(y|x)P(x)$$

$$P(x, y) = P(y, x)$$

**Assumes  
x and y  
independent!**

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Understanding Likelihood

How do we do this sort of maths for a general problem?

- Use Bayes' theorem:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

- Therefore:

$$P(\text{hypothesis} | \text{data}) = \frac{P(\text{data} | \text{hypothesis})P(\text{hypothesis})}{P(\text{data})}$$

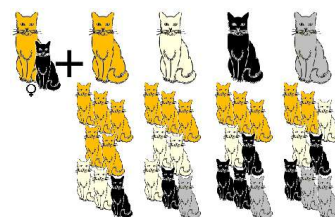
Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Understanding Likelihood

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

Therefore:



$$P(\text{father}_{\text{color}} | \text{kitten}_{\text{color}}) = \frac{P(\text{kitten}_{\text{color}} | \text{father}_{\text{color}})P(\text{father}_{\text{color}})}{P(\text{kitten}_{\text{color}})}$$

$$P(\text{father}_{\text{cream}} | \text{kitten}_{\text{black}}) = \frac{1/8 \times 1/4}{7/32} = \frac{1}{7}$$

$$P(\text{father}_{\text{black}} | \text{kitten}_{\text{black}}) = \frac{3/8 \times 1/4}{7/32} = \frac{3}{7}$$

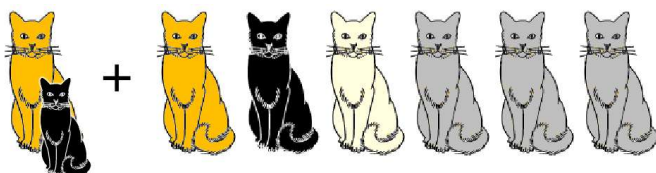
$$P(\text{father}_{\text{grey}} | \text{kitten}_{\text{black}}) = \frac{2/8 \times 1/4}{7/32} = \frac{2}{7}$$

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Understanding Likelihood

- What if the population of male cats is non-uniform?

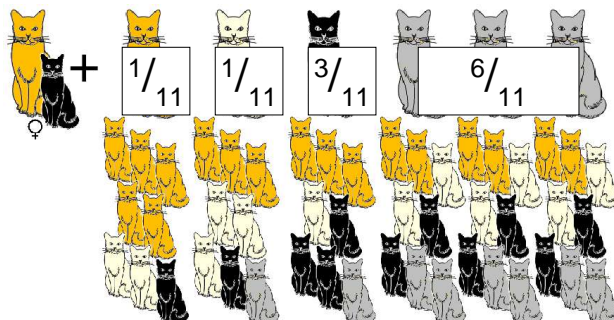


i.e.  $P(\text{father}_{\text{color}})$  is non-uniform

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Understanding Likelihood



Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Understanding Likelihood

$$P(\text{father}_{\text{color}}|\text{kitten}_{\text{color}}) = \frac{P(\text{kitten}_{\text{color}}|\text{father}_{\text{color}})P(\text{father}_{\text{color}})}{P(\text{kitten}_{\text{color}})}$$

$$P(\text{father}_{\text{orange}}|\text{kitten}_{\text{black}}) = \frac{1/8 \times 1/6}{11/40} = \frac{1}{11}$$

$$P(\text{father}_{\text{cream}}|\text{kitten}_{\text{black}}) = \frac{1/8 \times 1/6}{11/40} = \frac{1}{11}$$

$$P(\text{father}_{\text{black}}|\text{kitten}_{\text{black}}) = \frac{3/8 \times 1/6}{11/40} = \frac{3}{11}$$

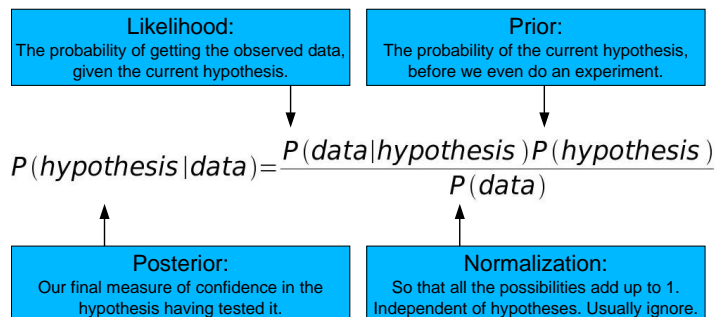
$$P(\text{father}_{\text{grey}}|\text{kitten}_{\text{black}}) = \frac{2/8 \times 3/6}{11/40} = \frac{6}{11}$$

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Maximum Likelihood

## Understanding Likelihood

- The elements of Bayes' theorem have names:

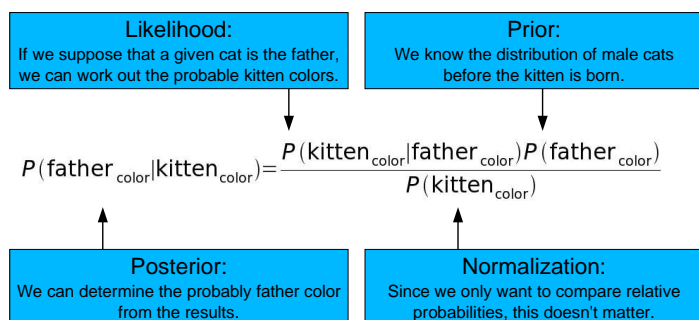


Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Maximum Likelihood

## Understanding Likelihood

- As applied to the cats:



Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Maximum Likelihood

## Likelihood and Crystallography

- How do we apply this to crystallography?
  - Hypothesis: The observed X-ray data arises from a molecule which is roughly homologous to this known molecule in this orientation in the cell. (Molecular replacement – how probable is a given model)
  - Hypothesis: The position of this atom (and its neighbors) better explains the X-ray data when moved in this direction. (Refinement – what is the relative probability of two very similar models. Includes heavy-atom refinement.)
- Each hypothesis leads to a set of predicted structure factors:  $F_c(\mathbf{h})$ . How well these explain the observed  $|F_{\text{obs}}(\mathbf{h})|$  determines the likelihood.

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Maximum Likelihood

## Likelihood and Crystallography

- For most purposes, we treat each reflection as an independent observation. Therefore we can consult each reflection separately to determine how well it agrees with the model. Then, we multiply all the resulting likelihoods together.
- Problem: the product of 10,000s of small numbers gives an underflow on a computer.**
- Solution: Take the log of all the likelihoods and sum them.**

$$\sum_i \log(x_i) = \log\left(\prod_i x_i\right)$$

Usually minimize  $-\log(\text{likelihood})$  because it is +ve.

## Likelihood and Crystallography

- Each hypothesis leads to a set of predicted structure factors:  $F_c(\mathbf{h})$ . How well these explain the observed  $|F_{\text{obs}}(\mathbf{h})|$  determines the likelihood.
- But: we have a continuum of hypotheses. We can rotate an MR model or move a refinement atom continuously to improve the model.
- We refine the parameters of the model (e.g. rotation of MR model, position of refinement atoms) in order to best explain the observed data, i.e. to give the highest value of the likelihood, hence:

**Maximum Likelihood**

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Maximum Likelihood

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Maximum Likelihood

## Likelihood and Crystallography

- But actually we want to maximize the posterior.  
e.g. in refinement:
  - Prior gives the probability of the model on the basis of its agreement with stereo-chemical restraints.
  - Likelihood gives the probability of the model on the basis of the observed X-ray data.
- If we just maximize the likelihood, we get lousy geometry.
- But people call it 'maximum likelihood' anyway.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

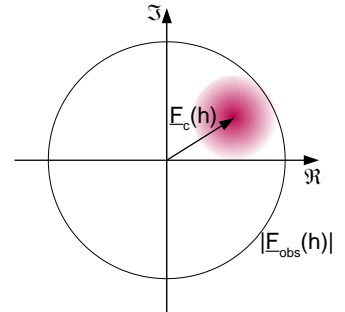
Sienna/Maximum Likelihood

## Likelihood and Crystallography

- Each hypothesis leads to a set of predicted structure factors:  $E_c(\mathbf{h})$ . How well these explain the observed  $|E_{obs}(\mathbf{h})|$  determines the likelihood.

- Note: To calculate a probability we must also estimate the error associated with the  $E_c(\mathbf{h})$ .

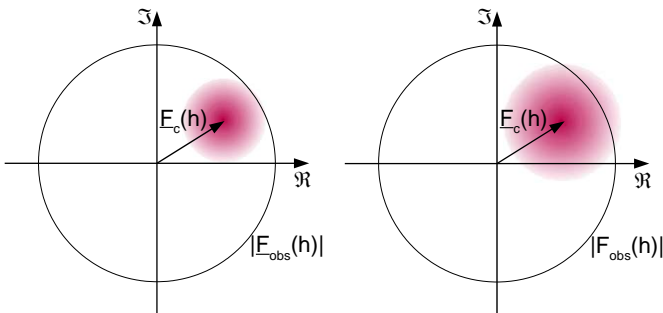
- The error estimation is a vital part of the model or hypothesis.



Sienna/Maximum Likelihood

## Likelihood and Crystallography

- How do we estimate the errors? Surely as the error estimate increases, the model always becomes a better description of the data?

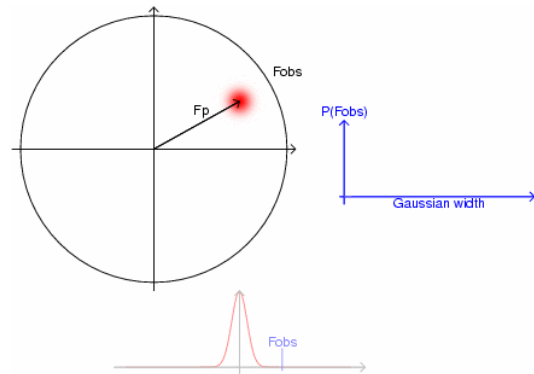


Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Likelihood and Crystallography

- No, the likelihood favors appropriate noise levels:

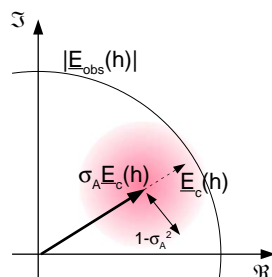


Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

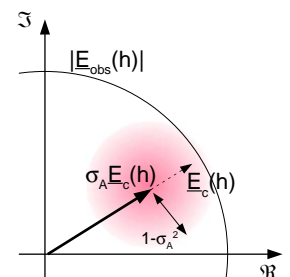
## Likelihood and Crystallography

- Error estimation is in terms of a parameter  $\sigma_A$ , where  $\sigma_A$  is the fraction of the normalised structure factor  $E_c(\mathbf{h})$  which is correct, and  $(1-\sigma_A^2)$  is the variance of the noise signal.
- Typically estimated as a function of resolution.
- Read (1986) Acta Cryst A42, 140-149



## Likelihood and Crystallography

- We can calculate the Likelihood Function for  $E_{obs}$  given  $E_c$ :



$$P(E_{obs}|E_c) \propto \exp\left(-\frac{|E_{obs} - \sigma_A E_c|^2}{\epsilon(1-\sigma_A^2)}\right)$$

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

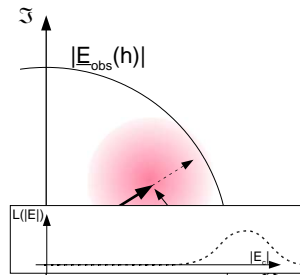
Sienna/Maximum Likelihood

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Likelihood and Crystallography

- But we don't know  $E_{obs}$  !
  - The data are the observed magnitudes:  $|E_{obs}|$
- We want  $P(\text{data}|\text{model})$
- Therefore, sum (integrate) the likelihood over all the unknown phases: Rice fn  
(i.e. eliminate nuisance variable)



$$P(|E_{obs}||E_c) \propto \exp\left(\frac{|E_{obs}|^2 + \sigma_a^2 |E_c|^2}{\epsilon(1 - \sigma_a^2)}\right) I_0\left(\frac{2|E_{obs}|\sigma_a |E_c|}{\epsilon(1 - \sigma_a^2)}\right)$$

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Likelihood and Crystallography

Steps:

- Construct a model, with some parameters: e.g.
  - MR: Rotation R, Translation T, Error term  $\sigma_A$
  - Refinement: Coords  $x_i$ , Temp factors  $B_i$ , Error term  $\sigma_A$
- Refine parameters R, T /  $x_i, B_i, \sigma_A$  to maximize the likelihood using the known **magnitudes**.
- Then use the resulting probability function for the **phases** to calculate an electron density map.
  - Programs will output ML map coefficients.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Likelihood and crystallography

Other details: Molecular replacement

- Programs will also use a likelihood function for unpositioned models to rank rotation function results.
- More complex likelihood functions allow combination of information from multiple fragments, even when relative position is unknown.
- See for example  
*Read (2001), Acta Cryst. D57, 1373-1382.*

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Likelihood and crystallography

Other details: Refinement

- Programs may also perform anisotropy correction, TLS refinement, bulk solvent correction. ML parameter refinement may be used to refine all of these parameters.
- Heavy atom refinement is similar, but is applied against multiple data sets simultaneously.
- See for example  
*Pannu, Murshudov, Dodson, Read, (1998) Acta Cryst. D54, 1285-1294.*

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood

## Likelihood and crystallography

Summary:

- Likelihood provides a tool for establishing the probability of a hypothesis.
- When data is weak, this is vital for describing our current state of knowledge.
- Direct benefits include improved models and weighted maps.
- Employed in:
  - phasing, MR, refinement, phase improvement, map interpretation.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Maximum Likelihood



## Overview of Crystallographic Structure Refinement

Lynn F. Ten Eyck

San Diego Supercomputer Center

Department of Pharmacology  
University of California, San Diego

IUCr Computing School, Siena, 2005

Lynn F. Ten Eyck

Crystallographic Refinement

## Outline

- 1 Introduction
- 2 Problem Statement
- 3 Optimization Methods
- 4 Description and Parameterization of Model
- 5 Implementation Strategy
- 6 Summary

Lynn F. Ten Eyck

Crystallographic Refinement

## Introduction

- The objective is to obtain the best estimate of what is in the crystal.
  - “Best” means a model that contains all essential elements
  - “Best” also means measures of certainty about what is or is not present
  - “Best” also means quantitative estimates of parameters and uncertainties in parameters
- The method is optimization of the fit of data calculated from the model to the observed data
  - Robust optimization
  - Sensitive optimization
  - Use optimized results to find problems with the model

Lynn F. Ten Eyck

Crystallographic Refinement

## Problem Statement

We are optimizing a function of our current model and our observations.

$$\Phi(\mathbf{p}) = \sum_{obs} w_i (ky_{i,o} - f(i, \mathbf{p}))^2$$

$$\frac{d}{d\mathbf{p}} \Phi(\mathbf{p}) = -2 \sum_{obs} w_i (ky_{i,o} - f(i, \mathbf{p})) \left( \frac{\partial f(i, \mathbf{p})}{\partial \mathbf{p}} \right)$$

Observations include both diffraction and chemistry, i.e. bond lengths and so forth.

Lynn F. Ten Eyck

Crystallographic Refinement

## Optimization Methods

All of the methods depend on the Taylor expansion, usually truncated to first order, but sometimes taken to second order

$$\Phi(\mathbf{p}) \approx \Phi_0 + \left\langle \left( \frac{\partial \Phi}{\partial \mathbf{p}} \right) | \Delta \mathbf{p} \right\rangle + \dots$$

- Linearize  $\mathbf{F}_c(\mathbf{h})$  in parameters and form normal equations
  - Solve by Conjugate Gradients
  - Solve by Linear Algebra
- Solve directly by non-linear function minimizer

Lynn F. Ten Eyck

Crystallographic Refinement

## Description of Model

- Description should take advantage of polymeric nature of biological macromolecules
- Description should **NOT** assume proteins are pure polymers!
  - Ligands, covalent and otherwise
  - Cross-links
  - A-B-A interactions
- Constraints are definitions, not refinable parameters
- Restraints are additional observation with weights

Lynn F. Ten Eyck

Crystallographic Refinement

## Implementation

- Break down into steps
  - Model  $\Rightarrow$  Map
  - Map  $\Rightarrow$  Structure Factors
  - Structure Factors + Model  $\Rightarrow$  Objective Function
  - Objective Function  $\Rightarrow$  Gradient wrt SF
  - Gradient wrt SF  $\Rightarrow$  Gradient wrt Map
  - Gradient wrt Map  $\Rightarrow$  Gradient wrt Model
- Expose each step for possible editing
- Especially note editing of gradients to implement constraints

## Summary

- Design for Change
- Keep overall concept SIMPLE
- Efficiency only matters in places
  - In those places, it matters a LOT
- DESIGN FOR VALIDATION



## Refinement II - Modern Developments

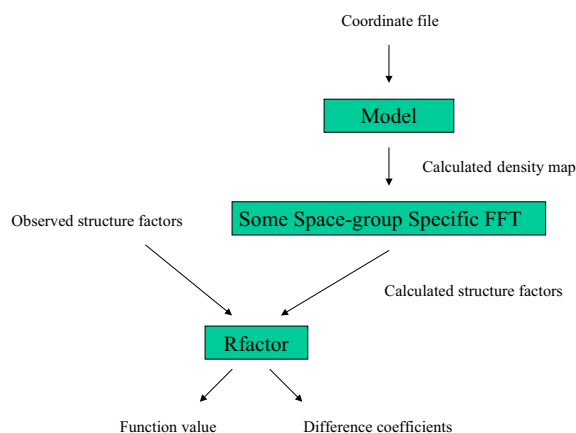
Dale E. Tronrud  
Howard Hughes Medical Institute  
University of Oregon

### Relevant Design Aspects

- The first popular reciprocal space refinement program was PROLSQ from Konnert and Hendrickson.
- This was a monolithic program.
  - To explore your own ideas of refinement you had to understand and modify their code.
- Lynn went the opposite way: a collection of programs coordinated by a scripting language.
  - One advantage of this design is that individual programs can be replaced with better implementations without worrying about the rest of the programs.
  - Another advantage is that one has access to the data stream between TNT programs, which allows one to add enhancements to without modifying or even trying to understand our programs.

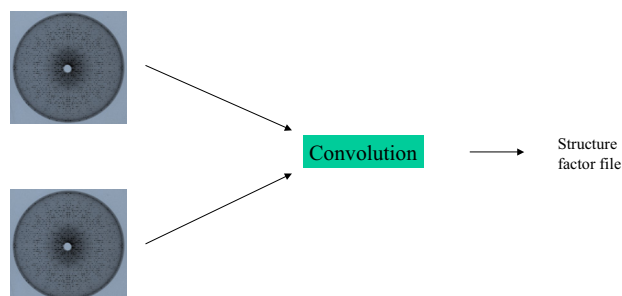
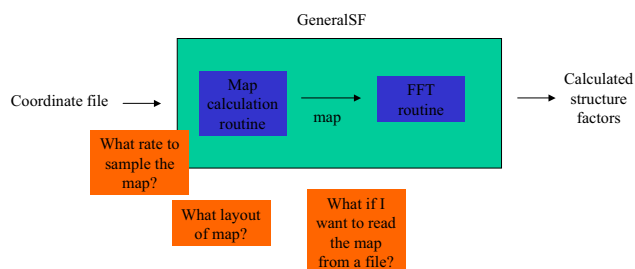
### TNT Refinement Package

- Designed and partially implemented by Lynn Ten Eyck in the late 1970's
- I have been working on it since 1981.
- We started distributing copies to other labs around 1984.
- It is used as the refinement engine in Buster/TNT from Gerard Bricogne's group.

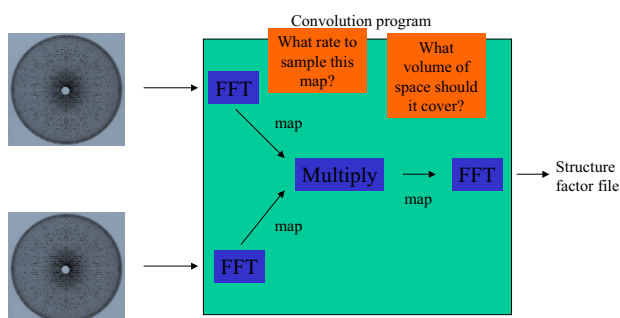


So, we write a single program for structure factor calculation

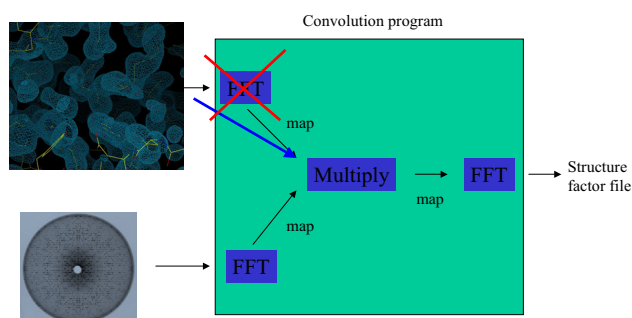
A more complicated example of the same problem



A more complicated example of the same problem

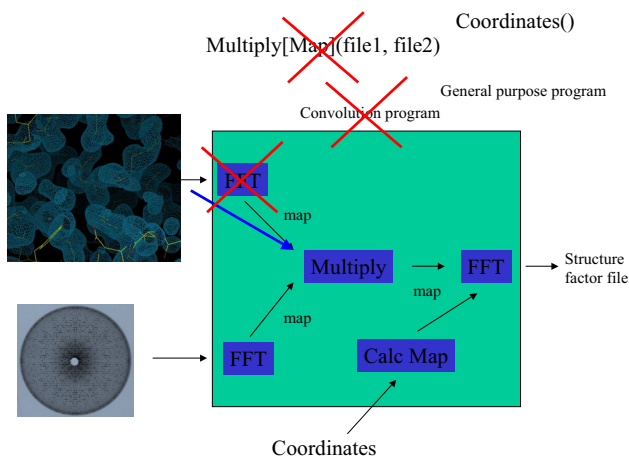


A more complicated example of the same problem



## The Solution

- Change the programming model.
    - Instead of programming serial operations, go to an event driven model.
  - The user isn't asking the program to do all these steps. The program should begin with the user's request.
    - I would like a structure factor file which contains the results of multiplying in real space the maps which correspond to the data in these two files.
  - The program should formalize this request and pass it to a routine (class, object, module, function, subroutine, whatever) whose job it is to fulfill requests.
  - The "request satisfier" will unfold the top layer of the request, and compose new requests for the data it needs.
  - It then recursively calls itself to acquire the data it needs.
- 
- The map multiplier knows that it needs two maps to multiply.
    - It generates a request for its first argument.
      - It knows that it needs an asymmetric unit filled with density. The sampling rate is more of a problem.
        - The resolution limit will be higher than the arguments. My code assumes that the two maps will have the same resolution so I double the sampling rate of the requests maps. This map should be oversampled by a factor of four.
      - The script for this request is simply "File1".
    - It sends its request for its first argument to yet another incarnation of the "request satisfier".
  - This incarnation sees that it has a request for a map but the source is a structure factor file. It calls the map calculating FFT and passes on the request.
  - The FFT knows that it needs structure factors and creates a request for them and passes that request, along with the script "File1" to yet another incarnation of the "request satisfier".
  - The satisfier sees that it is being requested to return structure factors from a structure factor file, so it does so. Finally the space group and resolution limit is known.
  - The FFT now knows the space group and resolution limit so it can decide on a sampling rate for the map and its layout.
  - It calculates the desired map and returns it to the satisfier who returns it to the map multiplier.
- 
- I would like a structure factor file which contains the results of multiplying in real space the maps which correspond to the data in these two files.
    - (While the program knows at this point if each file is a map, or HKL's, the "request satisfier" does not.)
  - The program rephrases this request as: Give me the set of structure factors that result from this calculation:
    - Multiply[Map](File1, File2)
  - The request is passed to the "request satisfier"
  - The "request satisfier" sees that structure factors are requested but the script returns a map, so it passes the request to the structure factor calculating FFT for processing.
  - The FFT builds a request for a map based on its needs.
    - It needs the map to cover an asymmetric unit of the unit cell.
    - It needs the sampling rate to be at least twice the resolution of the map.
    - It doesn't know the space group or the resolution so its request must be **vague**.
    - The new request is passed to another incarnation of the "request satisfier".
  - This "satisfier" sees a request for a map to be derived from an operator that creates a map.
    - It fires up the code that implements Multiply[MAP] and passes it the request.
- 
- The map multiplier receives the map for its first argument and builds a request for the second.
    - The second map must have the same sampling rate as the first because the grid points must line up. The layout must also be the same. This request is more specific.
  - The second argument request is passed down the line and a map is returned.
  - This map may not fulfill the needs of the request. The suppliers of data try their best to match a request but sometimes it is impossible.
    - The map multiplier must check that maps it receives are suitable for its calculation and adjust accordingly.
  - The space group of the product map may differ from the arguments. The true space group must be passed back to the requestor.
  - The map multiplier multiplies the maps and return the result to the structure factor calculating FFT.
  - The FFT now learns the space group, resolution, and sampling rate of its map.
    - It must verify that it can work with this result.
    - If so, it does the FFT and returns the structure factors.
  - The user finally gets the structure factors they wanted!



## A Other Examples

- `Apply_Mask(File, Envelope(Triangle(Truncate(File))))`
  - This script applies Wang-style solvent flattening to the map, or structure factors from a file.
- `Multiply[SF](File, Conjg[SF](File))`
  - Calculates Patterson coefficients, a Patterson Map, or Patterson peaks depending on what type is requested.

## Problems

- This entire concept is **illegal** in Fortran 77!
  - Recursion is forbidden.
- Designing the ability to make sufficiently vague requests is an ongoing task.
  - For example, when asking for an asymmetric unit of peaks you need a map that has a little extra around each edge of the asymmetric unit.
- Some decisions are hard without more knowledge.
  - For example, when zero-truncating a density map the resolution of the resulting map will likely be greater than the unmodified map. But by how much?
    - It depends on how many grid points you zero.

## What next?

- Currently all the structure factor and map calculations in TNT are done with this scheme.
- I would like to expand this to other data types, such as atomic shifts, gradients and curvatures. With this I could restructure the package to simply answer the request "I would like a better model given this model".
- **Reality check:**
- I am reaching the limits of what I can do in this language. I am thinking in object oriented terms but my tools are the classic "stone knives and bear skins". It's probably time for a complete rewrite.



## On Minimization Targets and Algorithms

Dale E. Tronrud  
Howard Hughes Medical Institute  
Institute of Molecular Biology  
University of Oregon, USA  
<http://www.uoxray.uoregon.edu/dale/>

## Refinement and Rebuilding

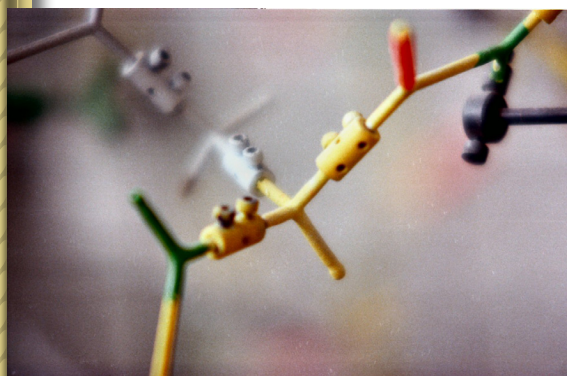
- Refinement and model rebuilding cannot be considered separately. Each is useless without the other.
- “Refinement” is a process where continuous changes are made to the model with consideration given to the observations.
- “Rebuilding” requires consideration of the context of each portion of the model with an eye towards making large changes, e.g. rotamer changes, peptide flips, as well as adding and deleting atoms.
- Rebuilding is usually manually controlled, although some protocols, such as arp/warp, are automated.

## What is Refinement?

- Refinement is the optimization of a function by changing the parameters of a model.
- What are the parameters?
  - Usually everyone agrees that the parameters are the position, B factor, and occupancy of each atom.
  - In most cases other parameters are added.
  - In some special cases different parameters are chosen.
- What is the function?
  - Maximum likelihood in REFMAC, CNS, BUSTER
  - Least squares in SHELX and TNT
  - Empirical energy in X-PLOR, CNS

- What minimization method?
  - Choice depends upon the function and the model.
  - Usually a variety of methods will be used on the same problem.
  - A variety of methods are used; several being used in the same project.
    - Simulated Annealing
    - Conjugate Gradient
    - Preconditioned Conjugate Gradient
    - Sparse Matrix
    - Full Matrix
    - . . .

## What Parameters to Refine?



## Elaborate Parameterization

- If one holds to the traditional parameterization of position and isotropic B's
  - High resolution and more precise lower resolution data will be wasted.
- The cost of elaboration is an increase in the total number of parameters.
  - Too many parameters leads to “overfitting” and poor refinement results.
  - Finding the right balance is difficult and will vary from case to case.
- The recent success stories are:
  - Torsion angle simulated annealing at low resolution
  - TLS anisotropic B's at midrange resolution

## Who's Got What?

- All packages can refine positions and isotropic B's and atomic occupancies.
- All packages can perform rigid body refinement.
- SHELLX, REFMAC, and Restrain have individual anisotropic B's.
- REFMAC has TLS anisotropic B's built as a “tree”.
- CNS has a “torsion-angle” parameterization which greatly improves simulated annealing refinement.
- Each program has many options to vary parameterization. Check the documentation for the options available to you.

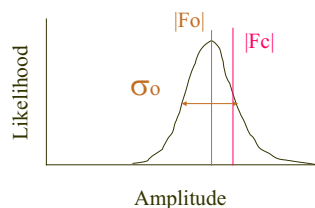
## What Function to Optimize?

## Maximum Likelihood

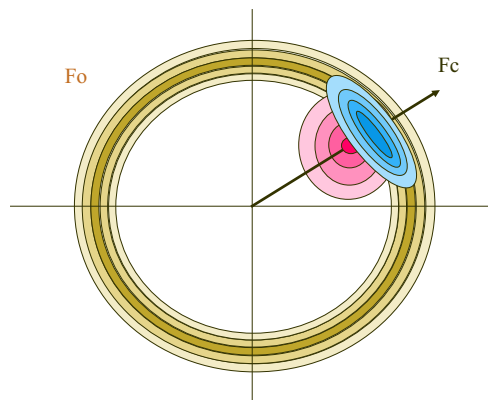
- Is a generalized statistical framework for optimizing models
  - Which means it is somewhat vague.
- The best set of parameters is the one for which the probability is the greatest that the experiment performed would result in the measured values.
- The character of the errors in your model are specifically built into the target function.
- If the errors in the experiment and the model obey certain assumptions Least Squares is the proper Maximum Likelihood method for the problem.
- Generally, the larger the errors the less applicable Least Squares becomes.

## Maximum Likelihood (cont.)

- Least Squares assumes that  $\frac{|F_o| - |F_c|}{\sigma_o}$  for all reflections obeys a Normal distribution with a mean of zero and a standard deviation of one.
- Least Squares view of the world:



## Maximum Likelihood's View



## Difficulties in Maximum Likelihood

- What is the character of the uncorrected errors in the model?
  - Existing programs assume the errors behave like randomly, and Normally distributed displacements of atomic positions and B factors.
  - Buster offers the option of non-uniform distribution of errors
    - It has a two state error model, where one part is treated in the usual way, but another part is identified only by a region of space and an elemental composition.
- How does one estimate the quantity of error in the model?
  - All ML programs use the agreement of the model to the test set to calibrate the error level.

## The Least-squares Function

- $Q_o(i)$  Observed quantity  $i$
- $\sigma_o^2(i)$  Observed variance of quantity  $i$
- $\mathbf{P}$  Parameters of a model
- $Q_c(i, \mathbf{p})$  Corresponding quantity inferred from the current model
- The best  $\mathbf{P}$  is that which minimizes

$$f = \sum_i \frac{(Q_o(i) - Q_c(i, \mathbf{p}))^2}{\sigma_i^2(i)}$$

## Different Classes of Observations

- This equation can easily handle observations of many different classes.

$$f = \sum_i \frac{1}{\sigma_i^2(i)} (Q_o(i) - Q_c(i, \mathbf{p}))^2 + \sum_i \frac{1}{\sigma_i^2(i)} (Q_o(i) - Q_c(i, \mathbf{p}))^2 + \dots$$

- Examples of classes are:
  - Structure factor amplitudes
  - Bond length and angles
  - Torsion angles and planarity
  - Non-bonded contacts
  - Stereochemical B factor correlation
  - MIR Phases
  - Noncrystallographic symmetry

## Major Limitation of this Equation

- The equation assumes that the observations are statistically independent. Often this is not the case.
  - Some programs use non-independent stereochemical restraint categories.
  - Many particular stereochemical targets are correlated.
  - The presence of noncrystallographic symmetry creates dependencies between some (many) reflections.
- This limitation also exists in all Maximum Likelihood implementations to date.

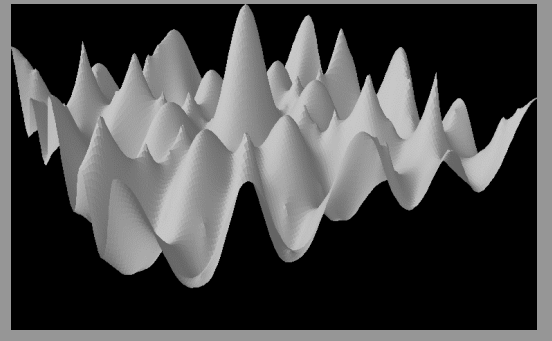
## Energy Minimization

- The best model with the one with the lowest energy.
- How does one calculate the “energy” of a model?
- How does the diffraction data become “energy”?
- How does one reconcile the instantaneous nature of energy with the time averaged nature of the diffraction data?
- Why bother when a statistically based method has answers to all these questions?

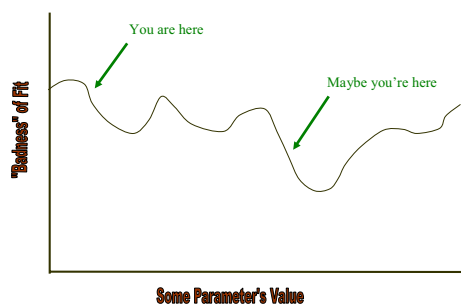
## How to Optimize the Function?

## Methods of Minimization

- Methods using no function derivatives
  - Simulated Annealing, Monte Carlo, Simplex, Metropolis
- Methods using first derivatives
  - Steepest Descent, Conjugate Gradient
- Methods using first and second derivatives
  - Full matrix, Block diagonal, Diagonal, Preconditioned Conjugate Gradient (, and Conjugate Gradient II)



## Simulated Annealing



## Full Matrix Minimization

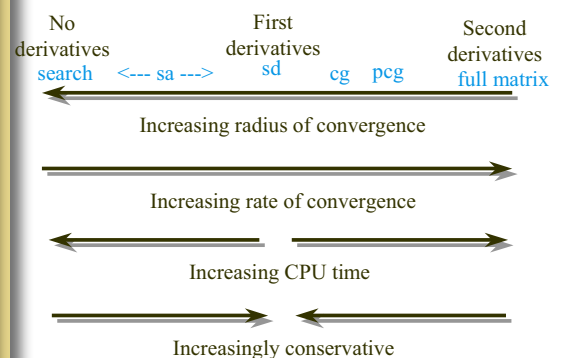
$$-\left[\frac{d^2 f(\mathbf{p})}{d\mathbf{p}d\mathbf{p}}\right]^{-1}_{\mathbf{p}=\mathbf{p}_0} \left[\frac{df(\mathbf{p})}{d\mathbf{p}}\right]_{\mathbf{p}=\mathbf{p}_0} = \mathbf{s}$$

- If the function is not quadratic
  - more than one cycle is required to reach the minimum.
  - an initial guess for the parameters is required.
- The second derivative matrix is huge
  - very time consuming to calculate and invert.
- The power of convergence is great.
- The radius of convergence is very poor.
- It absolutely requires an overdetermined problem.

## Approximations to Full Matrix

- Sparse Matrix
  - Only large matrix elements are used
- Block Diagonal
  - Assumes the parameters can be categorized
- Preconditioned Conjugate Gradient
  - Assumes all off diagonal elements are zero, but learns the truth from experience
- Gradient / Curvature
  - Assumes all off diagonal elements are zero, and is pig-headed about it.
- Conjugate Gradient
  - Assumes all diagonal elements are equal, but learns from experience
- Steepest Descent
  - Assumes all diagonal elements are equal

## The Minimization Continuum





## Who's Got What?

- Full matrix refinement is available
  - SHELLX, REFMAC, and RESTRAIN.
  - Full matrix programs usually allow for the exclusion of subsets of off-diagonal elements.
  - They will also offer a selection of means to approximate the inverse of the Normal matrix.
- Ignoring all off-diagonal elements leads to the diagonal approximation and preconditioned conjugate gradient.
  - TNT and BUSTER
- Ignoring the diagonal elements leads to steepest descent and conjugate gradient.
  - X-PLOR and CNS

## In Conclusion...

- This talk should help you to ask the right questions to find out how one refinement package differs from another.
- One must know the differences between the packages to make a rational decision about which to use in your project.
- One must also know the limitations of their package and what problems to watch for. No package is perfect.

## Computational aspects of the Rietveld Method

Needs for precise refinements and microstructural effects:  
Improvement of treatment of peak shapes, Rietveld algorithm, ...

Juan Rodríguez-Carvajal  
Laboratoire Léon Brillouin, (CEA-CNRS), CEA/ Saclay  
FRANCE



## Experimental powder pattern

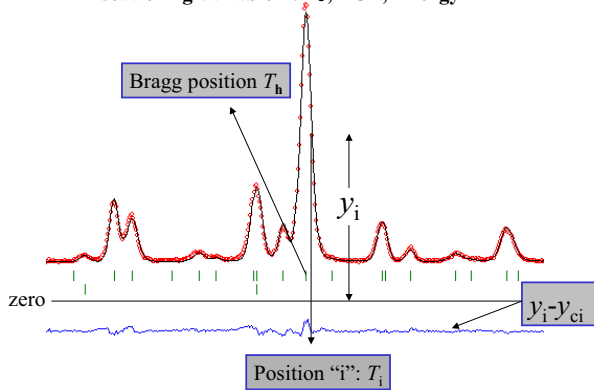
A powder diffraction pattern can be recorded in numerical form for a discrete set of scattering angles, times of flight or energies. We will refer to this scattering variable as :  $T$ . The experimental powder diffraction pattern is usually given as three arrays :

$$\{T_i, y_i, \sigma_i\}_{i=1,2,\dots,n}$$

The profile can be modelled using the calculated counts:  $y_{ci}$  at the  $i$ th step by summing the contribution from neighbouring Bragg reflections plus the background.



Powder diffraction profile:  
scattering variable  $T$ :  $2\theta$ , TOF, Energy



## The calculated profile of powder diffraction patterns

$$y_{ci} = \sum_{\{h\}} I_h \Omega(T_i - T_h) + b_i$$

$I_h = I_h(\beta_I)$  Contains structural information: atom positions, magnetic moments, etc

$\Omega = \Omega(x_h, \beta_P)$  Contains micro-structural information: instr. resolution, defects, crystallite size...

$b_i = b_i(\beta_B)$  Background: noise, incoherent scattering diffuse scattering, ...



## The calculated profile of powder diffraction patterns

$$y_{ci} = \sum_{\{h\}} I_h \Omega(T_i - T_h) + b_i$$

The symbol  $\{h\}$  means that the sum is extended only to those reflections contributing to the channel " $i$ ".

This should be taken into account (resolution function of the diffractometer and sample broadening) before doing the actual calculation of the profile intensity.

This is the reason why some Rietveld programs are run in two steps



## Several phases ( $\phi = 1, n_\phi$ ) contributing to the diffraction pattern

$$y_{ci} = \sum_{\phi} s_{\phi} \sum_{\{\phi h\}} I_{\phi, h} \Omega(T_i - T_{\phi, h}) + b_i$$

## Several phases ( $\phi = 1, n_\phi$ ) contributing to several ( $p = 1, n_p$ ) diffraction patterns

$$y_{ci}^p = \sum_{\phi} s_{\phi}^p \sum_{\{\phi h\}} I_{\phi, h}^p \Omega^p(T_i - T_{\phi, h}) + b_i^p$$



$$y_{ci} = \sum_{\{h\}} I_h \Omega(T_i - T_h) + b_i$$

$$I_h = S \{ L p O A C F^2 \}_h$$

Integrated intensities are proportional to the square of the structure factor  $F$ .

The factors are:

Scale Factor ( $S$ ), Lorentz-polarization ( $Lp$ ), preferred orientation ( $O$ ), absorption ( $A$ ), other “corrections” ( $C$ )

**The Structure Factor contains the structural parameters (isotropic case)**

$$F(\mathbf{h}) = \sum_{j=1}^n O_j f_j(h) T_j \sum_s \exp \left\{ 2\pi i \left[ \mathbf{h} \{ S | \mathbf{t} \}_s \mathbf{r}_j \right] \right\}$$

$$\mathbf{r}_j = (x_j, y_j, z_j) \quad (j = 1, 2, \dots, n)$$

$$T_j = \exp(-B_j \frac{\sin^2 \theta}{\lambda^2})$$



### Structural Parameters (simplest case)

$$\mathbf{r}_j = (x_j, y_j, z_j)$$

Atom positions (up to  $3n$  parameters)

$$O_j$$

Occupation factors (up to  $n-1$  parameters)

$$B_j$$

Isotropic displacement (temperature) factors (up to  $n$  parameters)

### Structural Parameters (complex cases)

As in the simplest case plus additional (or alternative) parameters:

- Anisotropic temperature (displacement) factors
- Anharmonic temperature factors
- Special form-factors (Symmetry adapted spherical harmonics), TLS for rigid molecules, etc.
- Magnetic moments, coefficients of Fourier components of magnetic moments, basis functions, etc.



### The Structure Factor in complex cases

$$F(\mathbf{h}) = \sum_{j=1}^n O_j f_j(h) T_j \sum_s g_j(\mathbf{h}_s) \exp \left\{ 2\pi i \left[ \mathbf{h} \{ S | \mathbf{t} \}_s \mathbf{r}_j \right] \right\}$$

$$\mathbf{h}_s = \begin{pmatrix} h \\ k \\ l \end{pmatrix}_s = S_s^T \begin{pmatrix} h \\ k \\ l \end{pmatrix} \quad (s = 1, 2, \dots, N_G)$$

$g_j(\mathbf{h}_s)$  Complex form factor of object  $j$   
Anisotropic DPs  
Anharmonic DPs

The peak shape function of powder diffraction patterns contains the **Profile Parameters**

$$\Omega(x_{hi}, \beta_p) = \Omega(T_i - T_h, \beta_p)$$

$$\Omega(x) = g(x) \otimes f(x) = \text{instrumental} \otimes \text{intrinsic profile}$$

$$\int_{-\infty}^{+\infty} \Omega(x) dx = 1$$

In most cases the observed peak shape is approximated by a linear combination of Voigt (or pseudo-Voigt) functions

$$\Omega(x) \approx \sum L(x) \otimes G(x) = \sum V(x)$$



## Properties of the Voigt function

$$V(x) = V_1(x) \otimes V_2(x)$$

The Voigt function has proven to be a very good experimental approximation in many cases

$$\beta_L = \beta_{1L} + \beta_{2L} \rightarrow \text{Lorentzian breadths simply have to be summed}$$

$$\beta_G^2 = \beta_{1G}^2 + \beta_{2G}^2 \rightarrow \text{Gaussian breadths have to be summed quadratically}$$

$$\beta_{fL} = \beta_{hL} - \beta_{gL} \rightarrow \text{Correction for instrumental broadening}$$

$$\beta_{fG}^2 = \beta_{hG}^2 - \beta_{gG}^2$$



However, the **Rietveld Method** can be easily extended by using, instead of the traditional  $\chi^2$  (least squares), another **Cost Function** to be minimised against the parameter vector  $\beta$

$$\text{Cost} = \sum_{i=1}^n F(\{y_i - y_{ci}(\beta)\})$$

$$\text{Cost} = -\log(\text{Likelihood})$$



The **Rietveld Method** consist of refining a crystal (and/or magnetic) structure by minimising the weighted squared difference between the observed and the calculated pattern against the parameter vector:  $\beta$

$$\chi^2 = \sum_{i=1}^n w_i \{y_i - y_{ci}(\beta)\}^2$$

$$w_i = \frac{1}{\sigma_i^2}$$

$\sigma_i^2$ : is the variance of the "observation"  $y_i$



## Least squares: Gauss-Newton (1)

Minimum necessary condition:  $\frac{\partial \chi^2}{\partial \beta} = 0$

A Taylor expansion of  $y_{ic}(\beta)$  around  $\beta_0$  allows the application of an iterative process. The shifts to be applied to the parameters at each cycle for improving  $\chi^2$  are obtained by solving a linear system of equations (normal equations)

$$\begin{aligned} A \delta \beta_0 &= b \\ A_{kl} &= \sum_i w_i \frac{\partial y_{ic}(\beta_0)}{\partial \beta_k} \frac{\partial y_{ic}(\beta_0)}{\partial \beta_l} \\ b_k &= \sum_i w_i (y_i - y_{ic}) \frac{\partial y_{ic}(\beta_0)}{\partial \beta_k} \end{aligned}$$



## Least squares: Gauss-Newton (2)

The shifts of the parameters obtained by solving the normal equations are added to the starting parameters giving rise to a new set

$$\beta_1 = \beta_0 + \delta \beta_0$$

The new parameters are considered as the starting ones in the next cycle and the process is repeated until a convergence criterion is satisfied. The variances of the adjusted parameters are calculated by the expression:

$$\begin{aligned} \sigma^2(\beta_k) &= (A^{-1})_{kk} \chi_v^2 \\ \chi_v^2 &= \frac{\chi^2}{N - P + C} \end{aligned}$$



## Least squares: a local optimisation method

- The least squares procedure provides (when it converges) the value of the parameters constituting the local minimum closest to the starting point
- A set of good starting values for all parameters is needed
- If the initial model is bad for some reasons the LSQ procedure will not converge, it may diverge.



## Needs for precise refinements and microstructural effects

Precise refinements can be done with confidence only if the intrinsic and instrumental peak shapes are properly approximated.

At present

⇒ The approximation of the **intrinsic profile** is mostly based in the **Voigt (or pseudo-Voigt) function**

⇒ The approximation of the **instrumental profile** is also based in the **Voigt function** for constant wavelength instruments

⇒ For TOF the **instrumental+intrinsic profile** is approximated by the convolution of a **Voigt function** with **back-to-back exponentials** or with the **Ikeda-Carpenter function**.



## Example: General $2\theta$ dependence of the instrumental broadening (determined by a standard sample)

$$H_{hG}^2 = (U_f + (1 - \xi_f)^2 D_{fST}^2(\mathbf{a}_D)) \tan^2 \theta + \frac{I_{fG}}{\cos^2 \theta} + H_{gG}^2$$

$$H_{hL} = (X_f + \xi_f D_{fST}(\mathbf{a}_D)) \tan \theta + \frac{[Y_f + F_f(\mathbf{a}_S)]}{\cos \theta} + H_{gL}$$

The Gaussian and Lorentzian components of the instrumental Voigt function are interpolated between empirically determined values. If needed, axial divergence is convoluted numerically with the resulting profile.



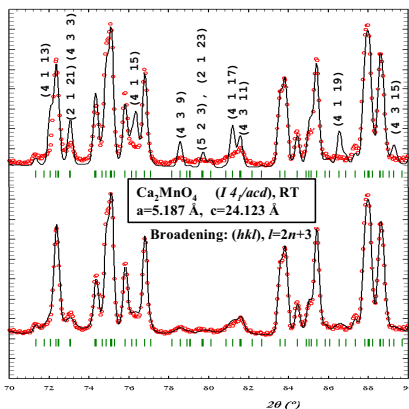
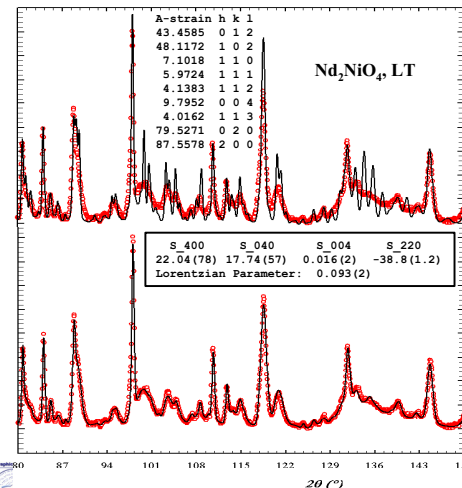
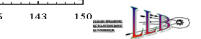
## Microstructural effects and peak shapes (Rietveld)

Recent developments:

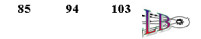
⇒ **Anisotropic peak broadening** (strain/size effects):  
quartic forms in  $hkl$  (dislocations, micro-twinning, composition fluctuations)  
spherical harmonics (complex size/microstrain effects)  
special reflections (stacking faults, antiphase domains, polytypes)

⇒  **$hkl$ -dependent shifts with respect to Bragg positions:**  
special reflections, quartic forms, ...

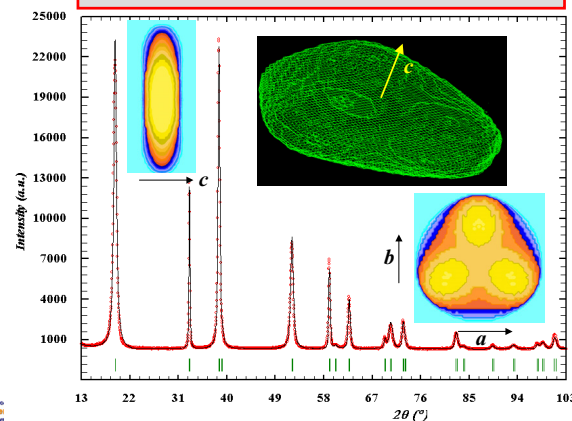
⇒ **New sample profiles:** Linear combination of pseudo-Voigt functions to mimic log-normal and gamma size distributions (Popa *et al. J. Appl. Cryst* 35, 2002, 338-346)



Selective size broadening observed by neutron diffraction at room temperature (3T2, LLB) for superstructure reflections in  $\text{Ca}_2\text{MnO}_4$  (top) Size parameter fixed to zero. (bottom) Single size parameter according to the rule  $(hkl), l=2n+3$ .



## Size broadening in $\text{Ni}(\text{OH})_2$



## Problems when modeling the peak shape, a real case: low resolution neutron powder diffractometers

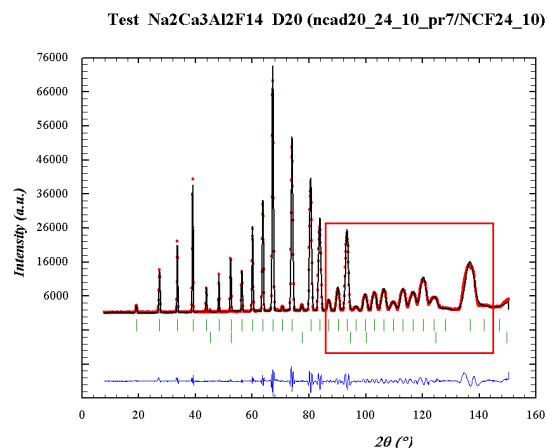
D20 at ILL:

A diffraction pattern can be collected in less than a second!

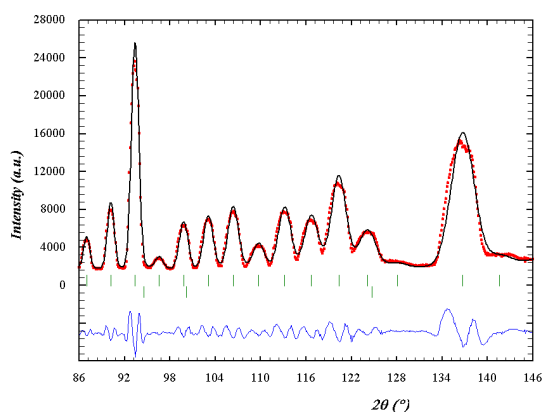
Large graphite monochromator with a quite low take-off angle ( $2\theta_M \approx 40^\circ$ )

This implies that at high angle the peaks are broad and have a strange peak shape (that can be reproduced quite precisely by ray tracing or Monte Carlo simulations of the instrument !)

ILL Consortium on Crystallographic Computing  
Siena 2009: Crystallographic Computing School



Test Na<sub>2</sub>Ca<sub>3</sub>Al<sub>2</sub>F<sub>14</sub> D20 (ncad20\_24\_10\_pr7/NCF24\_10)



The inability to model peak shapes properly introduces a “systematic error” in the data treatment affecting the structural parameters and the estimation of their uncertainties

How can we solve this problem?

ILL Consortium on Crystallographic Computing  
Siena 2009: Crystallographic Computing School



## Future developments for Rietveld analysis: the treatment of the peak shape

Further step on complexity:

- ⇒ Fundamental parameters approach
- ⇒ Numerical instrumental profile (when needed)
- ⇒ Local convolution with analytical sample profile using FFT or interpolated direct convolution

$$\Omega(x) = FT^{-1} [G(t)F(t)] = g(x) \otimes f(x)$$

This is partially performed (with analytical functions) in the CCSL based code at ISIS and in FullProf/GSAS for the TOF case:  $V(x) \otimes IK(x)$

In TOPAS the fundamental parameters approach is fully implemented

ILL Consortium on Crystallographic Computing  
Siena 2009: Crystallographic Computing School



## Future developments for Rietveld analysis: the treatment of the peak shape

- ⇒ Different components of both instrumental and sample profile functions are just multiplied in the Fourier space.
- ⇒ The global  $G(t)$  may be provided in the instrumental resolution file in different forms, depending if it can be approximated by analytical functions or not.

$$G(t) = G_1(t)G_2(t)G_3(t)...$$

$$F(t) = F_1(t)F_2(t)F_3(t)...$$

This procedure is faster than the direct convolution using numerical integration when the number of points per profile is greater than  $\sim 64$ .

ILL Consortium on Crystallographic Computing  
Siena 2009: Crystallographic Computing School



## The core of the algorithm used in the Rietveld Method

## Skeleton of the Rietveld algorithm

Calculations in a single cycle for all patterns

```
do n_pat=1,n_patt
  if(affpat(n_pat) == 0) cycle
  Select case(xunit(n_pat))
    Case("2theta")
      Call calc_pattern_2theta(n_pat)
    Case("TOF")
      Call calc_pattern_TOF(n_pat)
    Case("Energy")
      Call calc_pattern_Ed(n_pat)
    . . . . .
  End Select
end do
```



## The Rietveld algorithm: (do over points/reflections)

Subroutine calc\_pattern\_TOF(n\_pat)

```
.....
DO i=1,npts(n_pat)
  ini=code_contribution(i,n_pat,"ini")
  fin=code_contribution(i,n_pat,"fin")
  IF(iprev <= fin) THEN
    DO j=iprev,fin
      CALL calcul_tof(j,n_pat)
    END DO
    iprev=MAX(iprev,fin+1)
  END IF
  CALL summat_tof(i,n_pat,ini,fin)
END DO
return
End Subroutine calc_patterns_TOF
```



## The Rietveld algorithm: calculation for each reflection

SUBROUTINE calcul\_tof(nn,n\_pat)

```
.....
!Calculate contribution of micro-structure
CALL strain (n_pat,nn,h,iph,dst,ss,dvv)
CALL sizef (n_pat,nn,h,iph,dsiz,ss,dvs)
CALL shifhkl(n_pat,nn,h,iph,shv,ss,dshv)
!Calculate FWHM and so on ...
.....
!Different models to calculate the structure factors
Select case(Model_STF)
.....
case("Magnetic_reflection_IREPS")
  call calmag_bas(n_pat,nn,iph,h,...,fnn)
.....
End Select
.....
CALL correct_tof(n_pat,nn,iph,h,fnn,ider)
!-----Calculate and store part of derivatives
.....
RETURN
END SUBROUTINE calcul_tof
```



## The Rietveld algorithm: (do over points/reflections)

Subroutine calc\_patterns\_TOF(n\_pat)

```
.....
DO i=1,npts(n_pat)
  ini=code_contribution(i,n_pat,"ini")
  fin=code_contribution(i,n_pat,"fin")
  IF(iprev <= fin) THEN
    DO j=iprev,fin
      CALL calcul_tof(j,n_pat)
    END DO
    iprev=MAX(iprev,fin+1)
  END IF
  CALL summat_tof(i,n_pat,ini,fin)
END DO
return
End Subroutine calc_patterns_TOF
```



## The Rietveld algorithm: (do over reflections, make sums, derivatives, LSQ matrix )

SUBROUTINE summat\_tof(ipm,n\_pat,ini,fin)

```
.....
! Calculate Ycalc and its derivative w.r.t all parameters
DO i=ini,fin
  ! Profile calculation
  Select case (nprof(n_pat))
    case("pV-conv-exp")
      omega= tof_peak2(delta,ider)
    . . . . .
  End Select
  omegap= scale_Lp abs*omega
  yc(ipm,n_pat)=yc(ipm,n_pat)+omegap*ff(i,n_pat)*corr(i,n_pat)
  ! Loop over MAXS parameters for completing derivatives
  j=MOD(i,MaxOVERL)+1
  DO k=1,maxs
    deriv(k)=dersto(j,k)*der*omegap+deriv(k)
  END DO
END DO
! Derivatives w.r.t. background parameters
! Construction of the Least-squares Matrix and Vector
.....
END SUBROUTINE summat_tof
```





## The Rietveld algorithm: summary

```

Do for N_cycles
  Do for Patterns                               ⇐ may be done in parallel
    Do for points in Patterns
      Do for contributing reflections
        Calculate broadening w.r.t to IRF
        Calculate structure factors+derivatives
        Sum contributions (LSQ matrix + vector)
        calculate profile for current point and reflections
        contributing to it (convolution ⇒ neighbours needed)
        profile derivatives
      End do reflections
    End do points in Patterns
  End do Patterns
  Invert LSQ matrix and update the free parameters
  Tests for convergence (if convergence is reached exit!)
End do N_cycles
  
```



## May the Rietveld algorithm be improved?

New ideas are needed to improve the efficiency:

New data structures?  
 Store individual peak shapes?  
 Change the order of loops?  
 Modularise different parts of the calculations?

...

## The Rietveld algorithm in a context of increasing complexity

With the forthcoming high performance instruments, and increasing complexity, **we need an improvement of the algorithms for handling Rietveld refinements if we want to preserve interactivity.**

Options:

- ⇒ Develop small specialized Rietveld programs
- ⇒ Maintain the possibility of general treatment and improve the efficiency by making strong changes on the Rietveld codes.
- ⇒ A combination of both ...



## Conclusions and perspectives

The increasing complexity of instruments and the necessity of better refinements call for collaborative teams for improving the existing software and develop new tools.

A list of tasks and priorities is needed to undertake a rational software development.

This opens new opportunities for young people wishing to dedicate their scientific activities to  
 Crystallographic Computing.



## Statistical Treatment of Uncertainties

Dale E. Tronrud  
Howard Hughes Medical Institute  
Institute of Molecular Biology  
University of Oregon, USA  
<http://www.uoxray.uoregon.edu/dale/>

## Introduction

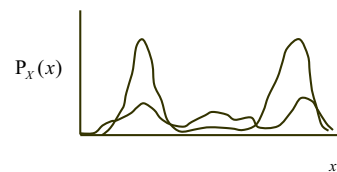
- There has been much discussion of uncertainty in this workshop. All this talk simply means that the topic is critical to everything that we do.
- Unfortunately many people, and most of the users of our software, would prefer that everything is clear cut and certain.
- The world is filled with uncertainty and true understanding requires us to know the limits of our knowledge.
- My hope here is to clarify some of the terms and issues in this area. Terms tend to be used without clear definition and even the experts confuse them quite often.

## Major Topics in Uncertainty

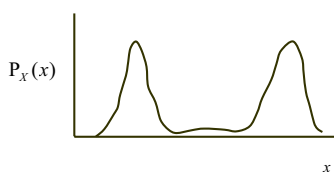
- There are three (ok, maybe four) major topics in this area
- Probability Distribution Functions -- PDF's
- Bayesian Parameter Estimation
- Maximum Likelihood -- ML
- Least-Squares

## Probability Distribution Functions (PDFs)

- Every quantity has an uncertainty. We like to say the uncertainty is plus or minus some amount but this is not sufficiently descriptive.
- With a PDF a probability is assigned to every conceivable value the quantity could "really" be.



Probability of the quantity  $X$  as a function of  $x$



- People would rather deal with something simpler than this entire distribution.
- The basic characteristics are
  - The most probable value
  - The expectation value (also known as the "mean", "best", or "centroid")
  - The standard deviation (also known as "sigma")
- There are others that are less used
  - Skew, Kurtosis, and Entropy
- For the Normal distribution the most probable value is the mean and sigma is what we are used to.

## Calculations in the presence of uncertainty

- If you have a quantity  $X$  what is the uncertainty of  $2X$ ?
- Actually you have to go to the PDF to find out. The rule is

$$P_{2X}(x) = P_X(x/2)$$

- Once you have created the PDF for  $2X$  you can derive things like the "best" value.
- For linear functions the most probable value and "best" value transform with the variable. If you multiply  $X$  by 2 then the most probable and "best" values will also be multiplied by 2.
- This is not true for nonlinear functions.

$$P_{X^2}(x | x \geq 0) = P_X(\sqrt{x}) + P_X(-\sqrt{x})$$

$$P_{X^2}(x | x < 0) = 0$$

- When doing math, the key is to work on the PDF and then recalculate the characterizing values and not to try to calculate the mean of the transformed variable by transforming the mean.
- If you have a single measurement, you must come up with some idea of its uncertainty. If you have no idea how confident you can be in a value, it is useless.
  - To generate a PDF for a single measurement you must understand how that value was acquired.
  - For most measurements the uncertainty is a Normal distribution and the sigma would be determined from your knowledge of the instrument.
- If you have multiple measurements, transform them all and then calculate the mean.
- A major limitation in practice is that often we are unsure of the PDF.
  - This leads to the realization that each point in the PDF must be represented by a PDF itself. This makes my head hurt.

## Multiple Variables

- In any project there are many variables. While each has a PDF to represent its uncertainty, the uncertainties are often not independent of each other.
- For a reflection we usually consider the uncertainty in amplitude and phase to be independent, but that is not always the case. This leads to a two dimensional PDF



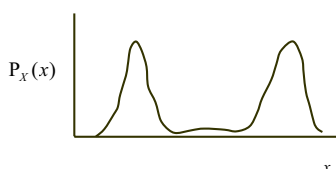
- Then again, the uncertainties of the amplitude and phase of a reflection are tied to its Friedel mate. This requires a four dimensional PDF.
- Then again, there is noncrystallographic symmetry. This requires an eight dimensional (or probably more) PDF.
- Let's face it, all reflections are tied to all others via crystal's contents. That requires a PDF that has a lot of dimensions.
- All interesting PDF's are multidimensional and usually they have an enormous number of dimensions.
- Finding some way to represent these PDF's is a problem yet to be solved.

## Getting a Handle on the Matter

- If you have a big, multidimensional PDF it might be true that there is only one peak and that peak looks like a Normal distribution.
- In that special case, the most probable value is the "best" value and you can gauge your uncertainty in that value.
  - For a 3 dimensional PDF this is not done with 3 sigmas.
  - You need a 3x3 covariance matrix.

$$\begin{pmatrix} \sigma_x^2 & \sigma_x \sigma_y r_{xy} & \sigma_x \sigma_z r_{xz} \\ \sigma_x \sigma_y r_{xy} & \sigma_y^2 & \sigma_y \sigma_z r_{yz} \\ \sigma_x \sigma_z r_{xz} & \sigma_y \sigma_z r_{yz} & \sigma_z^2 \end{pmatrix}$$

- Macromolecular crystallographers keep asking for a sigma for each parameter, but that would be uninformative without the correlation coefficients.



- The danger is that there are more peaks hiding in your PDF.
- In that case you could find both and calculate a covariance matrix for each as though the other doesn't exist.

## Bayesian Parameter Estimation

- Kevin has talked quite a bit in the last session about this topic and I don't see a need to repeat what he said.

$$P(h | d) = \frac{P(d | h)P(h | \text{knowledge})}{P(d | \text{knowledge})}$$

- Each of these probabilities is a multidimensional PDF.
  - In refinement they have an incomprehensible number of dimensions.
- Even if you could determine the probabilities for all possible hypothesis's you would have to search through them all to find the most probable, or integrate over them all for the "best" hypothesis.
- This is equivalent to the Global Minimum problem that has plagued us in refinement.
- No one has come up with a solution for this, for macromolecular refinement.
- The solution is to find a guess for the most probable hypothesis some other way, assume there can be only one, and then search for the most probable nearby hypothesis in the neighborhood.

## Maximum Likelihood

- Finding the most probable hypothesis in the Likelihood distribution, assuming there is only one peak, is the Maximum Likelihood method.
  - It finds the most probable, not necessarily the “best” hypothesis.
  - There may be whole worlds of stuff happening elsewhere in the likelihood distribution, but that is ignored.
- Bayesian Parameter Estimation is a rigorous and robust procedure, but is very hard for our problems.
- Maximum Likelihood is a massive simplification that allows us to bring in many of the ideas of BPE but reduces the problem to something very similar to what our old programs did.

## A Comparison of the Methods used in Macromolecular Refinement.

- There are three types of target functions that have been used in macromolecular refinement.
  - Energy Minimization
  - Least Squares
  - Maximum Likelihood

## Energy Minimization

- The best model with the one with the lowest energy.
- How does one calculate the “energy” of a model?
- How does the diffraction data become “energy”?
- How does one reconcile the instantaneous nature of energy with the time averaged nature of the diffraction data?
- Why bother when a statistically based method has answers to all these questions?

## The Least-squares Function

- $Q_o(i)$  Observed quantity  $i$
- $\sigma_o^2(i)$  Observed variance of quantity  $i$
- $\mathbf{p}$  Parameters of a model
- $Q_c(i, \mathbf{p})$  Corresponding quantity inferred from the current model
- The best  $\mathbf{p}$  is that which minimizes

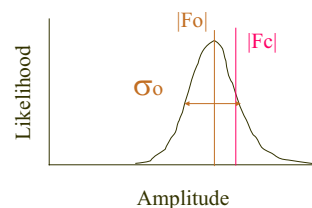
$$f = \sum_i \frac{(Q_o(i) - Q_c(i, \mathbf{p}))^2}{\sigma_o^2(i)}$$

## Major Limitation of this Equation

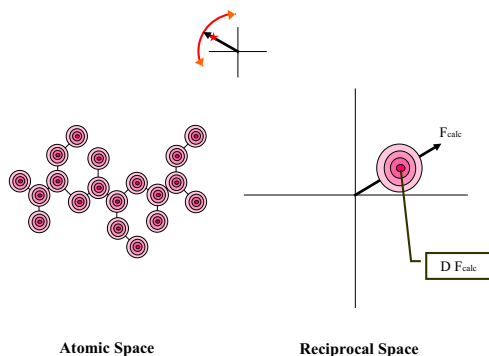
- The equation assumes that the observations are statistically independent. Often this is not the case.
  - Some programs use non-independent stereochemical restraint categories.
  - Many particular stereochemical targets are correlated.
  - The presence of noncrystallographic symmetry creates dependencies between some (many) reflections.
- This limitation also exists in all Maximum Likelihood implementations to date.

## Maximum Likelihood

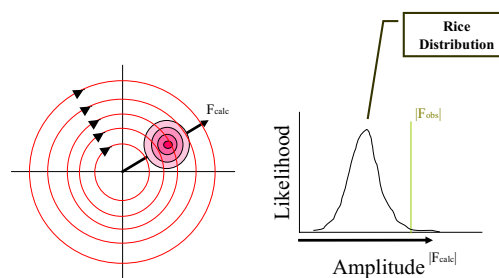
- Least Squares assumes that  $\frac{|F_o| - |F_c|}{\sigma_o}$  for all reflections obeys a Normal distribution with a mean of zero and a standard deviation of one.
- Least Squares view of the world:



## Is Structure Factor Calculation Hard?



## Getting to Amplitudes



## Difficulties in Maximum Likelihood

- What is the character of the uncorrected errors in the model?
  - Existing programs assume the errors behave like randomly, and with Normally distributed displacements of atomic positions and B factors.
  - Buster offers the option of non-uniform distribution of errors
    - It has a two state error model, where one part is treated in the usual way, but another part is identified only by a region of space and an elemental composition.
- How does one estimate the quantity of error in the model?
  - All ML programs use the agreement of the model to the test set to calibrate the error level.

## Programming pdCIF and Rietveld:

Brian H. Toby  
NIST Center for Neutron Research

## Motivation for Standardized Data Formats

Back in the dark ages of crystallography every program used its own data format.

- Electronic data communication was unusual and even then by sneakernet.

*Even then many crystallographers felt there must be a better way...*

## Data Grammars vs. Data Languages

A Data Grammar specifies how information will be formatted so that a computer program can interpret it.

- JCAMP-DX
- Spreadsheet (.csv)
- HDF
- STAR
- XML

## Talk Outline

- *Motivation/Goals*
- *Data Grammars vs. Data Languages*
- *CIF*
- *Informatics and the next generation of Rietveld software*
- *Programming writing & reading CIFs*

## Modern goals:

- Direct communication between instruments and data analysis tools
- Interoperability between programs
- Electronic communication of results
- Facile publication

*Productivity increases when computers function for scientists rather than the other way round!*

## XML syntax

```
<ObjCryst Date="2002-08-09T14:35:06">
  <Crystal Name="Alumina" SpaceGroup="R -3 c">
    <Par Refined="0" Min="1" Max="100" Name="a">4.76055</Par>
    <Par Refined="0" Min="1" Max="100" Name="b">4.76055</Par>
    <Par Refined="0" Min="1" Max="100" Name="c">12.9965</Par>
    <Par Refined="0" Min="28." Max="171." Name="alpha">90</Par>
    <Par Refined="0" Min="28." Max="171." Name="beta">90</Par>
    <Par Refined="0" Min="28." Max="171." Name="gamma">120</Par>
  </Crystal>
  <Atom Name="Al1" ScattPow="Al">
    <Par Name="x">0</Par><Par Name="y">0</Par><Par Name="z">0.3519</Par>
  </Atom>
  <Atom Name="O1" ScattPow="O">
    <Par Name="x">0.33333</Par><Par Name="y">0</Par>
    <Par Name="z">0.25</Par>
  </Atom>
</ObjCryst>
```

Annotations in the diagram:

- Nested Objects:** Points to the inner `<Crystal>` and `<Atom>` elements.
- Open-Delimiter:** Points to the opening `<` tag of an element.
- Closing-Delimiter:** Points to the closing `</>` tag of an element.
- Value:** Points to the text content between the delimiters, such as `4.76055` or `0`.

## STAR syntax (used in CIF)

data_alumina_example		Block header
_cell_length_a	4.766	
_cell_length_c	12.95	
_cell_angle_alpha	90.	
_symmetry_space_group_name_H-M	'R -3 c'	
loop_		Data name
_atom_site_label		Data Value
_atom_site_type_symbol		
_atom_site_symmetry_multiplicity		
_atom_site_fract_x		
_atom_site_fract_y		
_atom_site_fract_z		
Al1 Al	12 0.0 0.0 0.34	Table of data ("loop")
O1 O	18 0.33 0.0 0.25	

## Data Language:

- Built on a data grammar (usually)
- Provides rigorous definitions for each data value
- Establishes validation information (usually)
- Defines logical relationships between data items (optional)

**CIF: first comprehensive & interoperable data language for crystallography**

## Data languages are not new

C12		0.661000	1.000000	-0.175266	0.352517	0.950755	0.0		
0.012936		0.012232	0.018491	0.002923	0.008046	0.001071		0	1
C13		0.661000	1.000000	-0.055392	0.314579	0.925104	0.0		
0.010921		0.009871	0.014343	0.000394	0.005261	-0.001410		0	1
O1		0.577000	1.000000	0.414647	-0.009305	0.801699	0.0		
0.014915		0.010743	0.020270	0.002523	0.008903	-0.001350		0	1
O2		0.577000	1.000000	0.068052	0.227607	0.664516	0.0		
								0	1

3.2.4.1 Positional Parameters				
The positional parameter cards have FORMAT (A6,3X,6F9.0).				
Columns	Type 0	Type 1	Type 2	Type 3
1-6	Up to six alphanumeric characters centered in the six-place field			
7-9	—			
10-18	[Feature #1]	[Feature #1]	[Feature #1]	$x_0$ (Å, Cartesian)
19-27	[Feature #2]	[Feature #2]	[Feature #2]	$y_0$ (Å, Cartesian)
28-36	$x$ (fractional, crystal)	$x$ (Å, crystal)	$x$ (Å, Cartesian)	$r$ (Å, cylindrical)
37-45	$y$ (fractional, crystal)	$y$ (Å, crystal)	$y$ (Å, Cartesian)	$\phi$ (°, cylindrical)
46-54	$z$ (fractional, crystal)	$z$ (Å, crystal)	$z$ (Å, Cartesian)	$z$ (Å, cylindrical)
63	0	1	2	3

## What's so special about CIF?

**Each data item in CIF is defined in a computer-readable dictionary**

>3,000 defined terms (250+ pages in Int. Tabl.)

uses subset of STAR data grammar

>20 years of development effort (adopted by IUCr in 1990.)

_atom_site_aniso_U_11	
_atom_site_aniso_U_12	
_atom_site_aniso_U_13	
_atom_site_aniso_U_22	
_atom_site_aniso_U_23	
_atom_site_aniso_U_33	(numb)

These are the standard anisotropic atomic displacement components in ångströms squared which appear in the structure factor term:

$$T = \exp \left\{ -2\pi^2 \sum_i \left[ \sum_j (U^{ij} h_i h_j a_i^* a_j^*) \right] \right\}$$

$h$  = the Miller indices,  $a^*$  = the reciprocal-space cell lengths.  
The unique elements of the real symmetric matrix are entered by row.

Appears in list containing \_atom\_site\_aniso\_label. Related item(s): \_atom\_site\_aniso\_B\_ (conversion). [atom\_site]

## CIF has redefined small molecule publishing

*CIF is the uncontested standard for communication of structure factors & crystal structure results*

- Reduces errors in print: Journals can use structure validation software
- Bond distance & angle tables are generated directly from the CIF
- Required for IUCr Journals (*Acta Cryst.*, etc.)

**Impact on mm (via PDB) is probably even larger**

## How does CIF work:

### CIF Syntax

- Data names (tags) & values
- loop\_: links sets of data names & sets of values (Tables)

- Dictionary specifies:

- Definition
- Rules on allowed values
- Category
  - All data names in loop must be in same category
- Loop rules

- Specifies which data items can/must/cannot be looped
- Specifies logical connections between loops
- Specifies a unique item for each loop

_cell_length_a	4.766
_cell_length_c	12.95
_cell_angle_alpha	90.

loop_	
_atom_site_label	
_atom_site_type_symbol	
_atom_site_fract_x	
_atom_site_fract_y	
_atom_site_fract_z	
Al1 Al	0.0 0.0 0.34
O1 O	0.33 0.0 0.25



## CIF Dictionaries

***CIF definitions are developed by teams with widespread interests***

- Core -- fundamental & single-crystal terms
- mmCIF -- macromolecular
- pdCIF -- powder diffraction
- msCIF -- modulated structures
- imgCIF -- 2D images
- symCIF -- symmetry
- rhoCIF -- electron density
- diffCIF -- diffuse scattering (in progress)

## CIF Information Sources

- Formal specifications, see:  
<http://www.iucr.org/iucr-top/cif/>
- Also see templates & examples on Acta Cryst. Author's Guides
- International Tables Vol. G (today in Florence!)
- Developer's discussion list  
<http://www.iucr.org/iucr-top/lists/cif-developers/>

## Editorial comment:

*pdCIF is far more important than a mechanism for data interchange & review.*

- CIF has the potential to be the cornerstone of the next generation of Rietveld analysis software

## The two dialects of CIF Dictionaries

- In the course of defining CIF dictionaries, the mmCIF designers wanted more database structure than required for CIF initially.
  - Created a dictionary for defining dictionaries:
    - DDL (data definition language)
  - After inflicting many database structures into DDL v1.x, the mmCIF was written using DDL2.x
  - Programs that read dictionaries need to be aware of DDL1 vs DDL2 differences,
  - programs that only read/write CIFs do not.
  - Discussions on merging DDLs are underway

## CIF for Powder Diffraction (pdCIF)

***Universal data format for powder diffraction***

Goals beyond those of CIF:

- Accommodate all types of powder data
  - Flexibility in conflict with mmCIF
- Document experimental geometry, conditions
- Record “raw” and processed diffraction data & Rietveld fits

## The future of “Rietveld”

- Problems increasingly are more complex than can be accommodated with powder data at any resolution
  - Need incorporate many additional types of observations and constraints
- Characterize non-periodic aspects of crystal structures
  - Local order; stacking faults; defects

## The Next Generation of Data Fitting

- Codes should be modular, glued by a scripting language: customization
- Data modules can compute contributions to design matrix & least-squares vector against data & restraints
- Hard constraint modules can reduce parameters using GSAS or Finger-Prince approach
- Minimizer modules can develop & apply shifts from Hessian
- Cost function modules can keep parameters in bounds by adding to design matrix or Hessian

## CIF without STAR?

- CIF contains 20 years of informatics design efforts
- CIF is poor for large data structures
  - HDF is a portable data grammar for large data volumes
  - NeXus (HDF for scattering) not yet a complete data language
- XML is state-of-art ASCII data grammar
  - Again not a data language

### **CIF definitions can be transferred to other data grammars**

- Efforts to pair XML and CIF are underway (c.f. IUCr Florence, 2005)
- A marriage between CIF & NeXus would benefit everyone

## Personal Experiences in CIF Programming

- GSAS2CIF
  - Exports GSAS refinements in CIF
  - FillTemplate
    - Enter information into CIF templates (EXPGUI)
  - CIFSelect
    - Set [don't] publish flag in bond distances (EXPGUI)
- pdCIFplot
  - Plots Rietveld fits from CIF
- CIFEDIT
  - CIF validation & editor
- CMPR

## CIF is the control file for next generation data fitting

*CIF defines the basis for state-of-art refinement data objects*

*Where models cannot be described, CIF must expand*

- Data item descriptions are rigorous so that structure factors can be computed directly from the CIF
  - Modulated structures
  - Need more defect model descriptions
- Powder data can be simulated to match data items found in CIF
- With CIF additions PDF fitting becomes straight-forward

## Programming with CIF: Resources

- International Tables Volume G
- Open-source CIF parsers (see [www.iucr.org/iucr-top/cif/software/](http://www.iucr.org/iucr-top/cif/software/)):
  - CIFtbx 3.0 [Fortran]
  - Rutgers mmCIF lib [C]
  - CBFlib (used in RasMol) [C?]  
<http://www.bernstein-plus-sons.com/software/CBF/>
  - PyCifRW [Python]  
[www.ansto.gov.au/natfac/ANBF/CIF/](http://www.ansto.gov.au/natfac/ANBF/CIF/)
  - CIFIO in XTAL pkg [RATMAC=FORTRAN]  
[xtal.sf.net](http://xtal.sf.net)

## GSAS2CIF: Challenges

- Potentially complex data structures:
  - N ( $\leq 9$ ) phases
  - M ( $\leq 99$ ) data sets
  - $N \times M + 1$  CIF Blocks (or 1 if  $N=M=1$ )
- Reuse of author-entered information (metadata)
- Avoid use of CIF parser

Toby, B. H., Von Dreele, R. B., and Larson, A. C., "Reporting of Rietveld Results Using pdCIF: GSAS2CIF", *J. Appl. Cryst.* **36**, 1290 (2003)

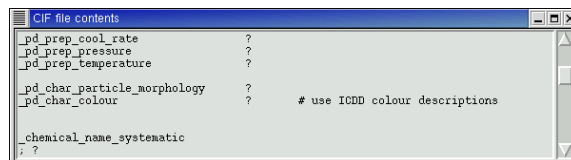
<http://www.ncnr.nist.gov/xtal/software/expgui/gsas2cif.html>

## GSAS2CIF: Solutions

- Divide *Acta Cryst.* template into sections
  1. Publication info
  2. Sample/characterization info (*need N copies*)
  3. Instrument/data collection info (*need M copies*)
    - Remove parameters “known” to GSAS
- CIF is generated by “quilting” together template sections with fit results
- Author-entered info (metadata) goes into template sections not into “final” CIF
  - Quick regeneration of new “final CIF”
  - Sharing of template sections between projects

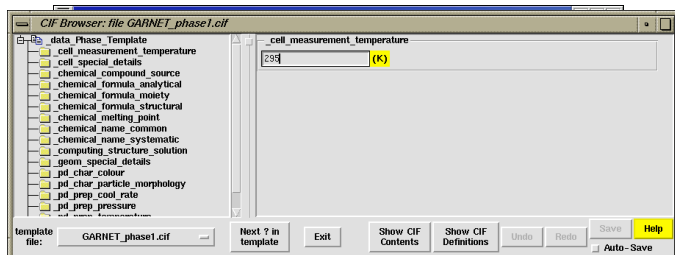
## GSAS2CIF GUI Tools: FillTemplate

Author must supply  
metadata -- entered  
into template



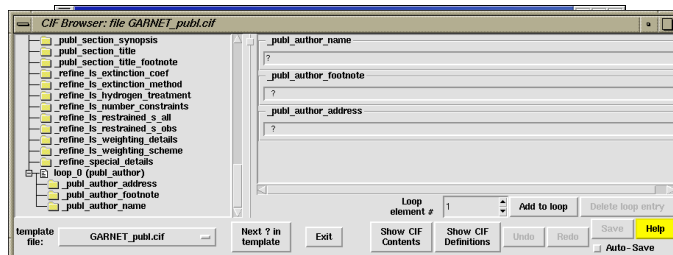
## GSAS2CIF GUI Tools: FillTemplate

Author must supply  
metadata -- entered  
into template



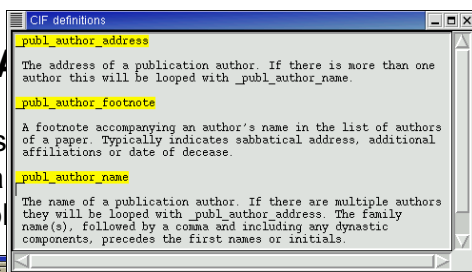
## GSAS2CIF GUI Tools: FillTemplate

Author must supply  
metadata -- entered  
into template



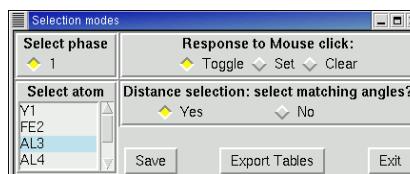
## GSAS2CIF

Author must supply  
metadata -- entered  
into template



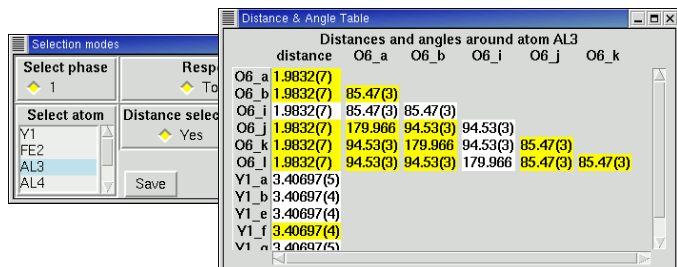
## GSAS2CIF GUI Tools: CIFSelect

Select distances & angles for publication  
– Keep flags in separate file



## GSAS2CIF GUI Tools: CIFSelect

Select distances & angles for publication  
– Keep flags in separate file



**Table 1**

pdCIF data items used for recording the powder diffraction dependent variable.

<code>_pd_meas_2theta_range_†</code>	Uncorrected $2\theta$ values with constant steps
<code>_pd_meas_2theta_scan</code>	Uncorrected $2\theta$ values, which may not have constant steps
<code>_pd_proc_2theta_range_†</code>	Calibration corrected $2\theta$ values with constant steps
<code>_pd_proc_2theta_corrected</code>	Calibration corrected $2\theta$ values, which may not have constant steps
<code>_pd_meas_time_of_flight</code>	Time for time-of-flight neutron diffraction measurements
<code>_pd_meas_position</code>	Linear detector position
<code>_pd_proc_energy_incident</code>	X-ray energy for energy-dispersive measurements
<code>_pd_proc_wavelength</code>	X-ray or neutron wavelength, when not constant
<code>_pd_proc_d_spacing</code>	$d$ spacing corresponding to an intensity value
<code>_pd_proc_recip_len_Q</code>	Momentum transfer ( $Q = 4\pi \sin \theta / \lambda$ ) for an intensity value

† The data names indicated as `_pd_XXXX_2theta_range_` actually correspond to three CIF data items, `_pd_XXXX_2theta_range_min`, `_pd_XXXX_2theta_range_max` and `_pd_XXXX_2theta_range_inc`, which define a range of equally spaced values.

## pdCIFplot: Challenge

- Change pdCIF from write-only to RW:  
Plot powder diffraction data/results from CIF
- Requirements
  - Select from many relevant data fields
  - Need Tcl/Tk CIF parser
- Task 1: tabulate possible data names for abscissa & ordinates

Toby, B. H., "Inspecting Rietveld Fits from pdCIF: pdCIFplot", *J. Appl. Cryst.* **36**, 1285 (2003)  
<http://www.ncnr.nist.gov/xtal/software/cif/pdCIFplot.html>

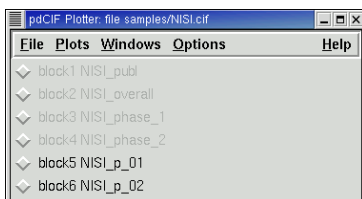
**Table 2**

pdCIF data items used for powder diffraction intensity values.

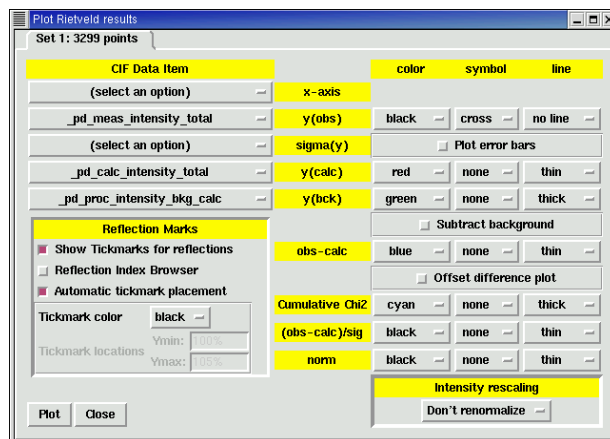
Observed intensities $y(\text{obs})$	Uncertainty values $\sigma_{y(\text{obs})}$
<code>_pd_meas_counts_total</code>	<code>_pd_meas_counts_total†</code>
<code>_pd_meas_intensity_total</code>	<code>_pd_meas_intensity_total†</code>
<code>_pd_proc_intensity_total</code>	<code>_pd_proc_intensity_total†</code>
<code>_pd_proc_intensity_net</code>	<code>_pd_proc_intensity_net†</code>
	<code>_pd_proc_ls_weight§</code>
† Standard uncertainty is the square-root of the counts for this data item.	
Background intensity $y(\text{bck})$	Calculated intensities $y(\text{calc})$
<code>_pd_meas_counts_background</code>	<code>_pd_calc_intensity_net</code>
<code>_pd_meas_counts_container</code>	<code>_pd_calc_intensity_total</code>
<code>_pd_meas_intensity_background</code>	
<code>_pd_meas_intensity_container</code>	
<code>_pd_proc_intensity_bkg_calc</code>	
<code>_pd_proc_intensity_bkg_fix</code>	

## Sequential GUI programming (ugh)

Select block (skipped if no choices)



## Specify plot contents



## Specify plot

Plot Rietveld results  
Set 1: 3299 points

**CIF Data Item**  
(select an option) **x-axis**

**pd\_meas\_intensity\_total** **y(obs)** black cross no line

(select an option) **sigma(y)** Plot error bars

**pd\_calc\_intensity\_total** **y(calc)** red none thin

**pd\_proc\_intensity\_bkg\_calc** **y(bck)** green none thick

**Reflection Marks**

☒ Show Tickmarks for reflections

☐ Reflection Index Browser

☒ Automatic tickmark placement

Tickmark color black

Tickmark locations Ymin: 100% Ymax: 105%

**obs-calc** blue none thin

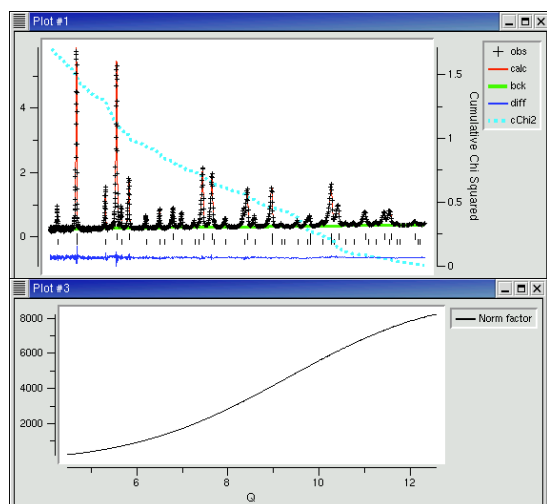
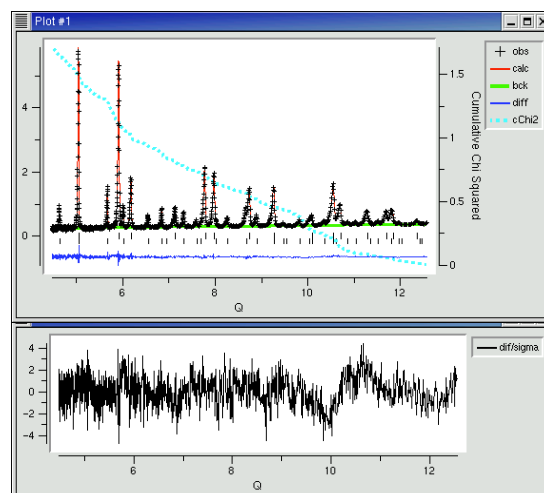
**Cumulative Chi2** cyan none thick

**(obs-calc)/sig** black none thin

**norm** black none thin

**Intensity rescaling**  
Don't renormalize

Plot Close





## Structure Comparison, Analysis and Validation

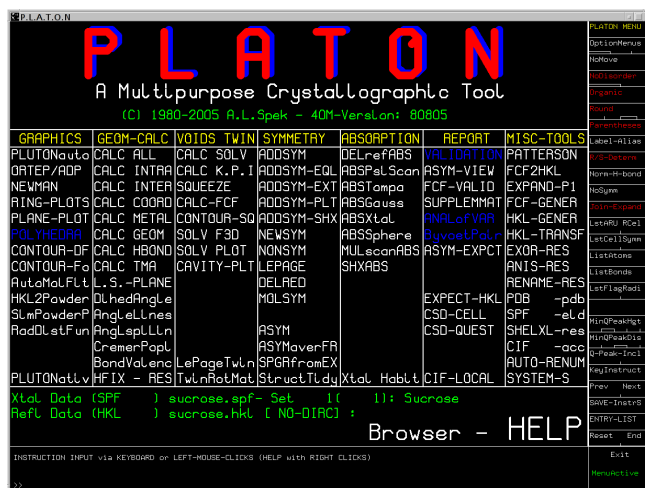
Ton Spek  
National Single Crystal Facility  
Utrecht University

## Overview

This lecture lists and discusses the various tools and descriptors that are available for the analysis and validation of a single crystal study as implemented in the PLATON program.

## Structure Analysis

- Analysis of the Intra-molecular Geometry
- Analysis of the Inter-molecular Geometry
- Analysis of the Coordination Geometry
- Bond Valence Model (Brown et al.)
- 'CALC ALL' - LISTING



## Intra-molecular Geometry

- Generation of the symmetry expanded **Connected Set** on the basis covalent radii plus a tolerance..
- Special tolerances are applied for certain types of X-Y bonds/contacts, either to include or avoid them.
- Residues grow from a starting atom by recursive spherical expansion.

## Intra-molecular Geometry

- Detection of residues and derivation of the Moiety formula, Z and Z'.
- Bond distances, Bond Angles, Torsion Angles.
- Automatic ring search, automatic search of planar parts in the structure

### Intra-Molecular (Continued)

- Determination of the hybridization, R/S assignments and ‘topology numbers’.
- Listing of the plane-plane and bond-plane angles.
- Ring puckering analysis (Cremer & Pople)

SP.LATON									
6-Membered Ring ( 2 )	(2) O(5) ←	C(11) ←	(2) C(2) ←	(2) C(3) ←	(4) C(4) ←	(5) C(5) ←			
	sp <sup>3</sup>	sp <sup>3</sup>	sp <sup>3</sup>	sp <sup>3</sup>	sp <sup>3</sup>	sp <sup>3</sup>			
Dev. (Ang)	0.1976(13)	-0.2107(9)	0.2394(12)	-0.2550(12)	0.2418(12)	-0.2191(12)			
Ce(1)-Reym-Par (Deg)	0.21(9)	0.95(9)	1.00(9)	0.21(9)	0.95(9)	1.00(9)			
C2(1)-Reym-Par (Deg)	110.75(9)	110.75(9)	110.75(9)	110.75(9)	110.75(9)	110.75(9)			
Ring Bond Angle(Deg)	115.36(9)	110.84(7)	111.02(7)	108.13(9)	110.82(10)	110.68(9)			
Tors(1)-U (Deg)	-54.91(12)	54.99(11)	-56.03(11)	56.27(11)	-54.87(12)	55.16(12)			
Ce(1)-U-Reym-Par (Deg)	110.53(12)	111.19(12)	110.52(12)	110.53(12)	111.19(12)	110.52(12)			
C2(1)-U-Reym-Par (Deg)	0.83(12)	1.12(12)	0.91(11)	0.83(12)	1.12(12)	0.91(11)			
Ring Bond Distance (Ang)	1.4108(14)	1.5346(13)	1.5198(16)	1.5257(15)	1.5258(17)	1.4386(19)			
Weighted Average Ring Bond Distance = 1.4937( 6.227) Ang. - NOTE: 1st ead. Internal, 2nd ead External.									
Weighted Average Abs. Torsion Ang. = 55.41( 5.26) Deg. see: e.g. Domenicano et al., Acta Cryst. (1975).									
Q(2) =	0.0498(12) Ang.	PH. (2) =	182.8(13) Deg						
Q(3) =	0.5542(12) Ang.								
Puckering Amplitude (Q) =	0.5965(12) Ang.	Theta =	5.12(12) Deg.	Phi =	182.8(13) Deg				

## Inter-Molecular

- Hydrogen Bonds (linear, bi- and trifurcated)
- Automatic analysis in terms of 1, 2 and 3-D networks (aggregates or cooperative)
- Search for pi-pi and C-H...pi interactions

[illegible]

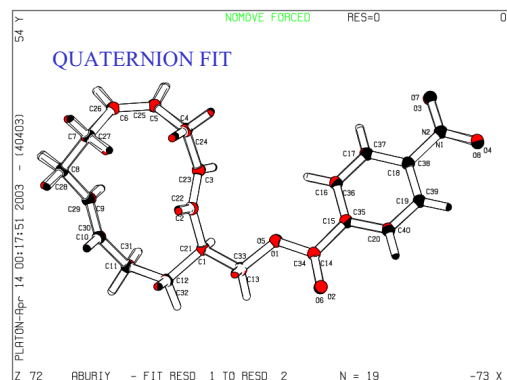
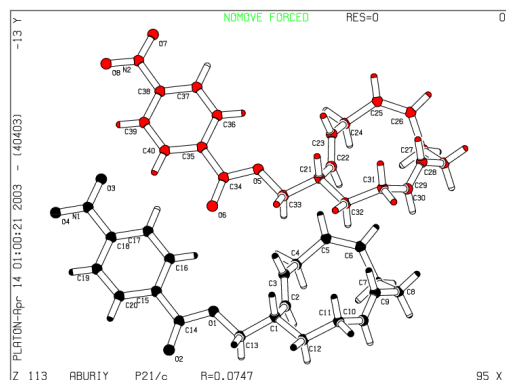
## Structure Comparison

- Quaternion Fitting
  - Modified version of A.L.Mackay (1984), A40,165-166 (Note: 180 degree singularity)
  - Alternative: S.K.Kearsley (1989), A45, 208-210.
- Comparison of Simulated Powder Patterns
- StructureTidy (Inorganics)

## QUATERNION FIT

- In many cases, an automatic molecule fit can be performed
- A) Identical atom numbering
- B) Sufficient number of Unique Atoms
- C) By manual picking of a few atom pairs





```
=====
Hofrit with Quaternion Method (A.L. Mackay, Acta Cryst. (1984), A40, 105-106)
=====

Fit Rotation axis about (Pseudo)axis [1,0,0] = 3.29 Degree
Direction Cosines with Orthogonal Cell 1,0,0 = 0.436719 0.830383 0.346553
Components in crystal system = 0.394049 1.000000 0.083576

Transf. Orthogonal Coord. Mol1 Orth. Coord. Mol2 with Resp. to C.G. Dist (Å)
O(1) 1.224 -0.879 -1.850 O(5) 1.198 -0.889 -1.868 0.033
O(2) 3.035 -1.529 -2.978 O(6) 3.008 -1.572 -2.964 0.052
N(1) -0.456 1.596 -7.411 N(2) -0.369 1.657 -7.412 0.106
C(1) 0.535 -1.424 0.352 C(21) 0.498 -1.430 0.337 0.041
C(2) 0.325 0.021 0.707 C(22) 0.253 0.000 0.695 0.076
C(3) -0.645 0.793 0.325 C(23) -0.743 0.741 0.291 0.116
C(4) -0.844 2.220 0.794 C(24) -0.972 2.175 0.700 0.146
C(5) -2.124 2.389 1.499 C(25) -2.203 2.340 1.516 0.095
C(6) -2.269 2.324 2.800 C(26) -2.291 2.305 2.823 0.037
C(7) -1.239 2.048 3.830 C(27) -1.178 2.089 3.790 0.084
C(8) -1.482 0.723 4.578 C(28) -1.303 0.782 4.602 0.162
C(9) -1.384 -0.460 3.664 C(29) -1.310 -0.424 3.725 0.102
... etc ...

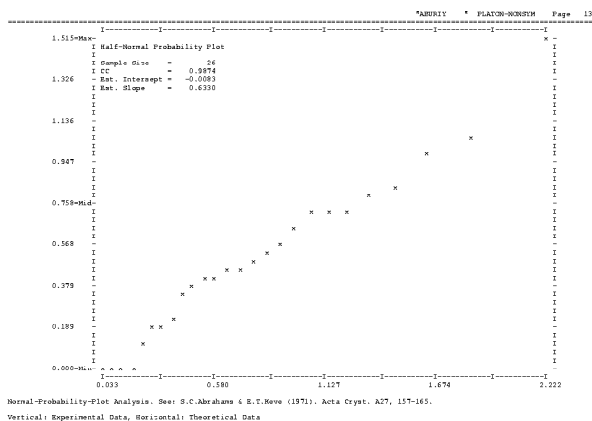
:: Weighted and Unit Weight RMS-Fit = 0.08729 0.08194 Angstrom

Cg1 0.946 0.234 0.592
Cg2 0.441 0.253 0.581
```

```
=====
*ABUR1
Comparison of the Bonds of the Fitted Residue
=====

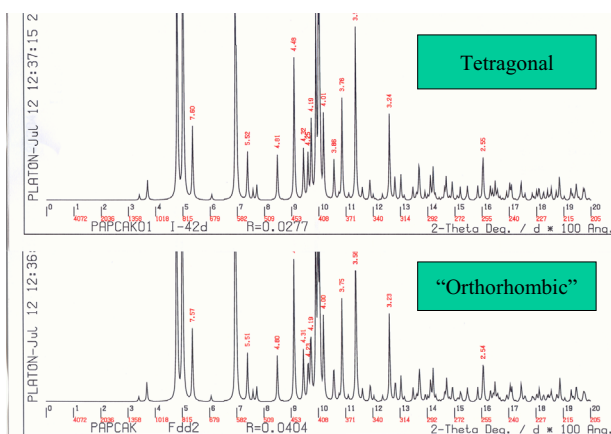
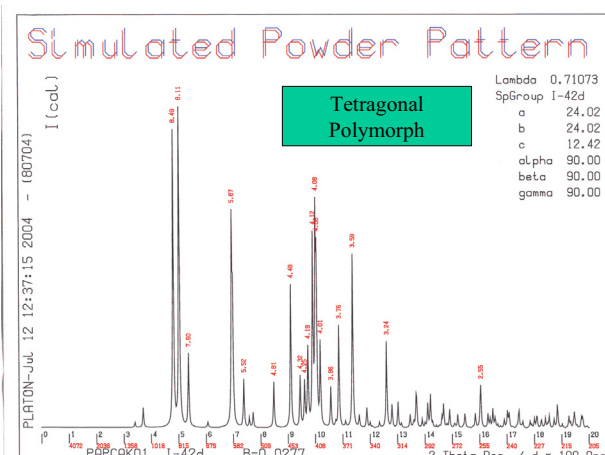
Read#1 Read#2 Dist#1 Dist#2 Diff Diff/Sig
O(1) -C(13) O(5) -C(33) 1.449(6) 1.440(6) 0.0090 1.0607
O(1) -C(14) O(5) -C(34) 1.326(6) 1.320(6) 0.0060 0.7071
O(2) -C(14) O(6) -C(34) 1.198(7) 1.198(6) 0 0
O(3) -N(1) O(7) -N(2) 1.225(7) 1.210(7) 0.0150 1.5152
O(4) -N(1) O(8) -N(2) 1.217(7) 1.210(7) 0.0070 0.7071
N(1) -C(18) N(2) -C(38) 1.477(8) 1.475(7) 0.0020 0.1881
C(1) -C(2) C(21) -C(22) 1.502(7) 1.494(7) 0.0080 0.8081
C(1) -C(12) C(21) -C(32) 1.523(7) 1.530(7) -0.0070 -0.7071
C(1) -C(13) C(21) -C(33) 1.504(7) 1.507(6) -0.0030 -0.3254
C(2) -C(3) C(22) -C(23) 1.297(7) 1.306(6) -0.0090 -0.9762
C(3) -C(4) C(23) -C(24) 1.503(7) 1.508(7) -0.0050 -0.5051
C(4) -C(5) C(24) -C(25) 1.491(8) 1.487(7) 0.0040 0.3763
C(5) -C(6) C(25) -C(26) 1.511(8) 1.511(8) 0 0
C(6) -C(7) C(26) -C(27) 1.483(8) 1.489(8) -0.0060 -0.5303
C(7) -C(8) C(27) -C(28) 1.537(8) 1.544(7) -0.0070 -0.6585
C(8) -C(9) C(28) -C(29) 1.496(8) 1.491(8) 0.0050 0.4419
C(9) -C(10) C(29) -C(30) 1.284(8) 1.284(8) 0 0
C(10) -C(11) C(30) -C(31) 1.496(8) 1.501(7) -0.0050 -0.4704
C(11) -C(12) C(31) -C(32) 1.520(8) 1.526(7) -0.0060 -0.5644
C(14) -C(15) C(34) -C(35) 1.483(7) 1.483(7) 0 0
C(15) -C(16) C(35) -C(36) 1.383(7) 1.385(6) -0.0020 -0.2169
C(15) -C(20) C(35) -C(40) 1.387(7) 1.383(7) 0.0040 0.4041
C(16) -C(17) C(36) -C(37) 1.375(7) 1.374(7) 0.0010 0.1010
C(17) -C(18) C(37) -C(38) 1.383(8) 1.374(7) 0.0090 0.8467
C(18) -C(19) C(38) -C(39) 1.374(7) 1.372(7) 0.0020 0.2020
C(19) -C(20) C(39) -C(40) 1.378(7) 1.374(7) 0.0040 0.4041

:: RMS Bond Fit = 0.0060 Ang.
```

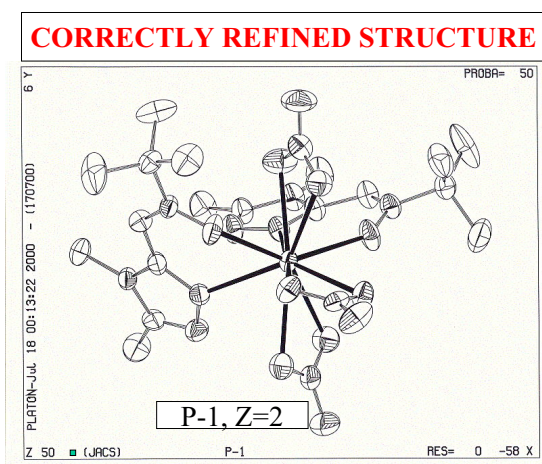
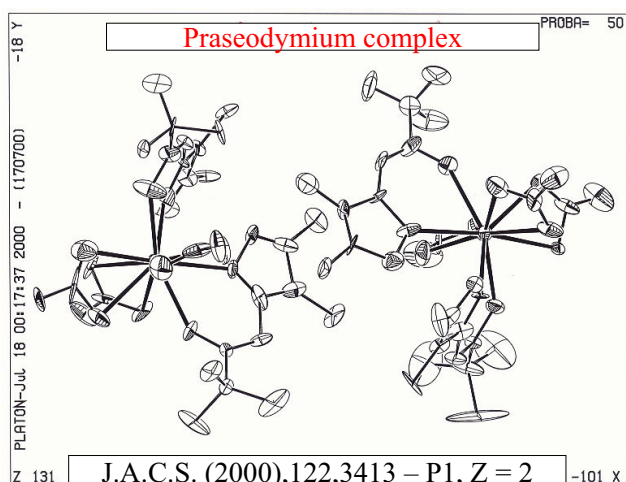


## Simulated Powder Patterns

- It is not always apparent that two crystal structures are identical. The assigned unit cell or space group can differ.
- Comparison of the associated calculated powder patterns should solve the issue.
- Example for the CSD:



- ## Absolute Structure



## STRUCTURE VALIDATION

Single crystal structure validation addresses three important questions:

- 1 – Is the reported information complete?
- 2 – What is the quality of the analysis?
- 3 – Is the Structure Correct?

## IUCR-CHECKCIF

### IUCR-TESTS:

- MISSING DATA, PROPER PROCEDURE, QUALITY

### PLATON TESTS:

- SYMMETRY, GEOMETRY, DISPLACEMENT PARAMETERS

### ALERT LEVELS:

- ALERT A - SERIOUS PROBLEM
- ALERT B - POTENTIALLY SERIOUS PROBLEM
- ALERT C - CHECK & EXPLAIN

## Problems Addressed by PLATON

- Missed Higher Space Group Symmetry
- Solvent Accessible Voids in the Structure
- Unusual Displacement Parameters
- Hirshfeld Rigid Bond test
- Miss-assigned Atom Type
- Population/Occupancy Parameters
- Mono Coordinated/Bonded Metals
- Isolated Atoms

## Problems Addressed by PLATON

- Too Many Hydrogen Atoms on an Atom
- Missing Hydrogen Atoms
- Valence & Hybridization
- Short Intra/Inter-Molecular Contacts
- O-H without Acceptor
- Unusual Bond Length/Angle
- CH<sub>3</sub> Moiety Geometry

## Validation with PLATON

- Details: [www.cryst.chem.uu.nl/platon](http://www.cryst.chem.uu.nl/platon)
- Driven by the file **CHECK.DEF** with criteria, ALERT messages and advice.
- Use: **platon -u structure.cif**
- Result on file: **structure.chk**
- Applicable on CIF's and CCDC-FDAT
- FCF-Valid: **platon -V structure.cif**

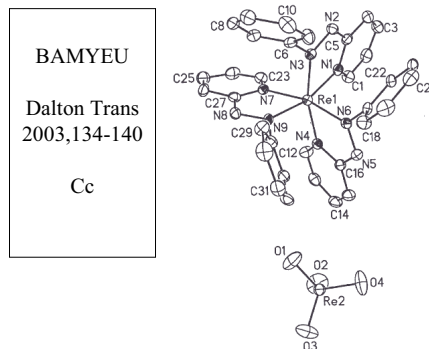
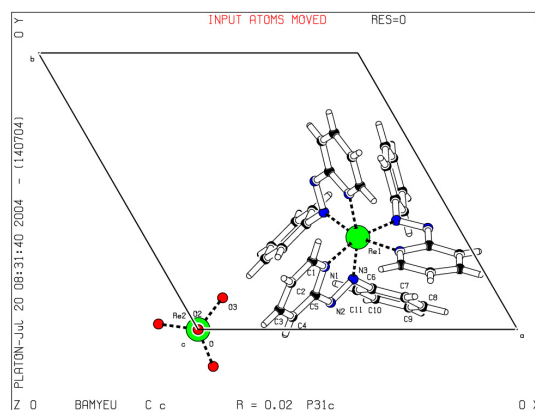
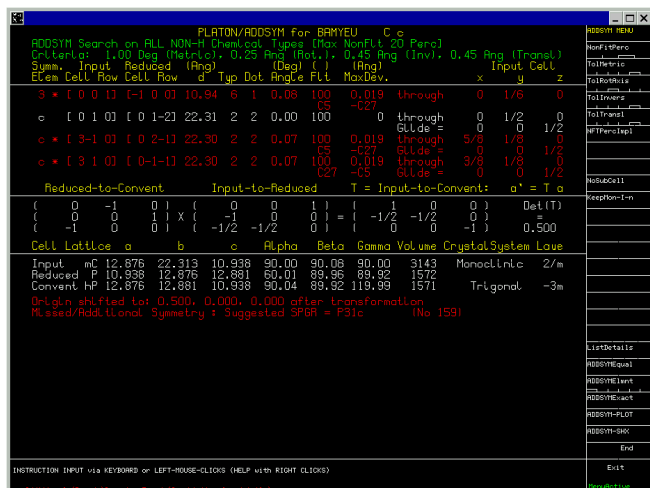
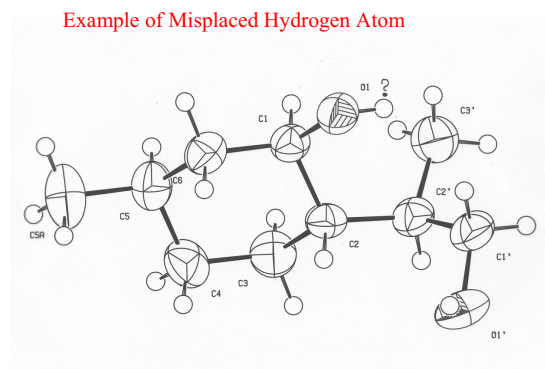


Fig. 4 A perspective view of  $[\text{Re}(\text{L})_3]\text{ReO}_4$ , 4a. The atoms are represented by their 30% thermal probability ellipsoids.



## Misoriented O-H

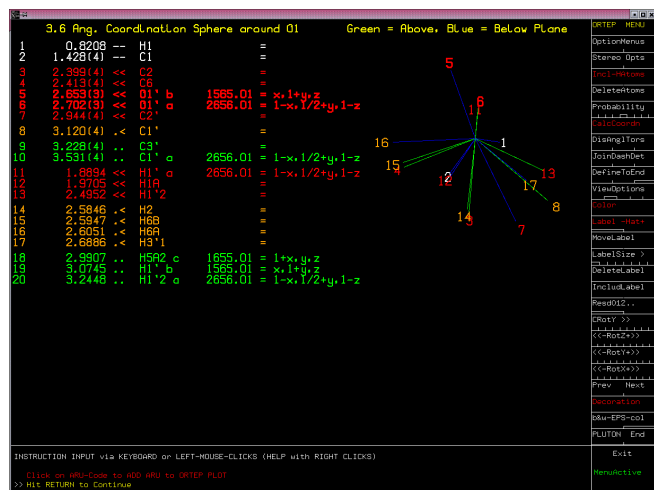
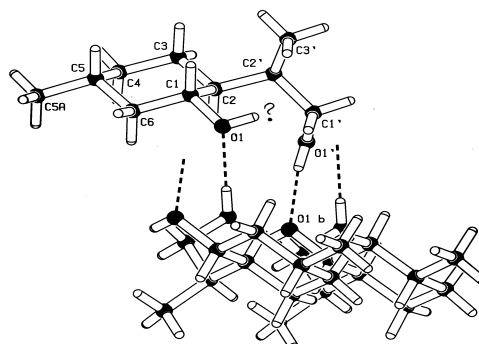
- The O-H moiety is generally, with very few exceptions, part of a D-H...A system.
- An investigation of structures in the CSD brings up many 'exceptions'.
- Closer analysis shows that misplacement of the O-H hydrogen atom is in general the cause.



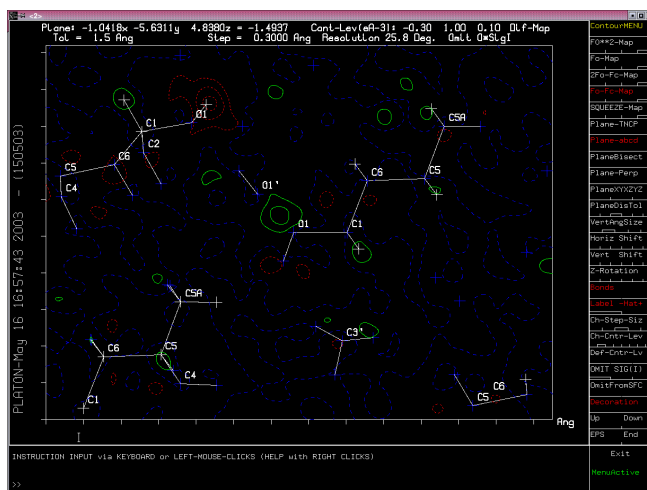
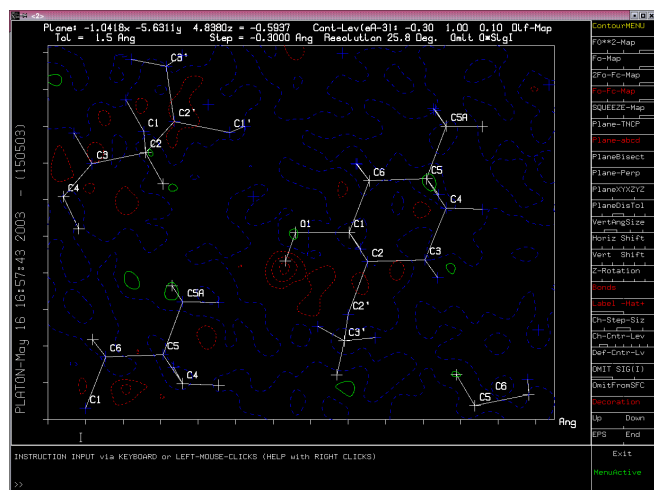
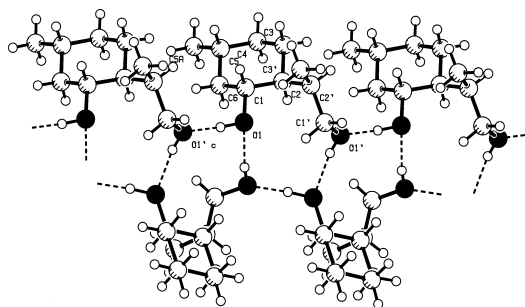
### Unsatisfactory Hydrogen Bond Network

```
# PLATON/CHECK run versus check.def version of 21/06/00                for entry:1
Data class: publ; Cif Data Type: CIF
# CELL      0.71069    8.571    6.466   9.850   90.00    106.78    90.00    522.69
# SpaceGroup P21
# MoleculeFormula C10 H20 O2
# Reported ?
# Sumformula C10 H20 O2 Rep: C10 H20 O2
# Mu          172.26[Calcl.] 172.26[Rep]
# Dx,gcm-3    1.094[Calcl.] 1.094[Rep]
#           2 [Calc.] 2 [Rep]
# Mu (m=1) = 0.074[Calcl.] 0.074[Rep]
# Calculated T limits Tmin=0.993 Tmax=0.997 Tmax=0.996
# Reported Lmax= 10 Kmax= 7 Lmax= 11 Nref= 1935 Th(max)= 25.76
# Calculated Hmax= 10 Kmax= 7 Lmax= 12 Nref= 1090(1991), Ratio= 1.78(0.97)
#           0 0 0 Wd=0.1870 U=0.721 Sigma=112
=====
>>> The Following ALERTS were generated <<<
420.ALERT B D-M without acceptor O(1) - H(1) ? <<<
048.ALERT C High R2 value Given 0.19
084.ALERT C High R2 value Given 0.19
084.ALERT C Inconsistent B value Given 0.19
142.ALERT C su on b-axis small or missing (x 100000) ... 30 Ang.
145.ALERT C su on beta small or missing (x 100000) ... 30 Deg.
145.ALERT C D-H short D-C H(2) H(12)? 1.93 Ang <<<
708.ALERT C D-H...A Calc 170.5(Sj) Rep 170.00, Dev 1.05 Sigma
O(1) 1.955 O(1) 1.555 2.646

1 ALERT Level B = Potentially Serious Problem
1 ALERT Level C = Check & Explain
```



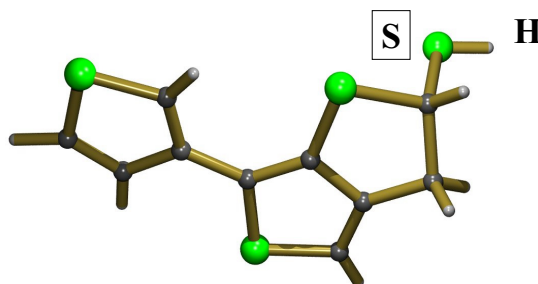
### Satisfactory Hydrogen Bond Network with new H-position



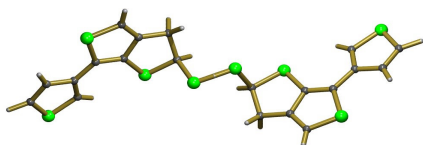
## Consult the CSD

- It is a good idea to always consult the CSD for previous reports on structures related to the one at hand.
- The statistics provided by VISTA (CCDC) can be very helpful for this.
- However, such an analysis often shows outliers. Many of those appear to be errors.

## Entry from the CAD



## But with Space Group Symmetry



=> Different structure with S-S Bond !

## Concluding Remarks

- Automatic Validation both ALERTS for potential errors and for interesting features in a structure to be discussed.
- Detailed analysis of intermolecular interactions appears often to be ignored in a service setting.



## Testing software

Harry Powell MRC-LMB

Testing has different purposes:

- do existing functions still work?
- do new functions work?
- does it give the customer what they want?
- is it an improvement over what existed before?
- performance testing

and different phases:

- alpha testing (core developers)
- beta testing (other developers + trusted users)
- release (the wider community)

Alpha testing is performed by the core developer(s) before any users see anything

- Uses standard input to give standard output
- tests new features (makes sure they don't break existing ones)
- Should get rid of all obvious bugs

Beta testing should be carried out by a small group of *intelligent* users:

- Once alpha testing is complete and no obvious bugs exist
- Makes sure features behave as expected in a non-sterile environment
- Reliable experienced users who will give full reports of failures (log files, circumstances, etc.)

Release is to the world at large ("real" users) and provides the most brutal testing

- you will enter the wonderful world of user support
- expect to get bug reports like "I pressed a button and it broke"
- users find "odd" bugs which arise from abuse of your carefully crafted software
- naïve users will find innovative ways of running (ruining?) your program

Reasons for testing existing software:

1. Want an identical result
2. Will accept a similar result
3. Want a different (better) result
4. Portability across platforms
5. Checking installation & performance

Types of testing:

1. background/batch/command-line
2. effect of different input
3. interactive *via* a GUI



You may want an *identical* result if:

- you have fixed an “unrelated” bug and don’t want to change the outcome
- you have tidied up the code (e.g. rewritten a routine)
- you have changed the optimization in compilation (e.g. from “-O0 g3” to “-O5 -funroll-loops”)
- you have used a different compiler (e.g. xlf instead of g77)
- you’ve *only* changed OS (e.g. rebooted from Linux to MS-Windows on the same box)
- you are running the program in different modes with the same input (e.g. batch mode or through a GUI)

You may accept a *similar* result if:

- you have just ported to a different chip (e.g. from PowerPC to i686)
- you are using a “random” seed
- your input is different
- you are using a different compiler

e.g. SGI Octane vs HP Alpha (autoindexing tetragonal lysozyme):

```
alf1_harry> diff alpha.lp irix_6.5_64.lp
186c186
< 20 306 tI 110.12 115.94 36.83 71.7 90.0 90.2
---
> 20 306 tI 110.12 115.95 36.83 71.7 90.0 90.2
188c188
< 18 204 oI 36.83 110.12 115.94 89.8 71.7 90.0
---
> 18 204 oI 36.83 110.12 115.95 89.8 71.7 90.0
223c223
< Initial cell (before refinement) is 77.8506 77.8506 36.8255
90.000 90.000 90.000
---
> Initial cell (before refinement) is 77.8507 77.8507 36.8255
90.000 90.000 90.000
```

You may want a *different (better)* result if:

- you have just spent six months improving an underlying algorithm
- you’ve implemented something new
- there are better traps for bad input
- you’ve been bug fixing

Batch testing (once set up) is easier, more reliable and less tedious than running a GUI

set up a shell script to

- run the program(s)
- check the output against a standard
- do the work while you do something more interesting

then expand the functionality as you realize you need it

Using a GUI usually means that you have to sit at a terminal and work through sets of examples and compare the answers (but with a scripted GUI (e.g. written in Tcl or Python) this can also be automated to some extent)...

Use a shell (csh or bash) or a scripting language?

csh (or tcsh) is the most common shell used by protein crystallographers

bash is most commonly used by computer scientists

zsh is a new tcsh-like shell which is becoming popular.

small molecule & powder crystallographers are often less familiar with shells

Largely a matter of personal choice, but bash’s syntax is a little more flexible and internal counters can be larger (but csh mutates less between platforms, and bash is missing on many SGIs).

For small scripts, a shell language is suitable, but for rigorous testing a proper scripting language may make further development easier (but remember James Holton’s Elves - 63,000 lines of csh).

```
#!/bin/bash -f
export IPMOSFLM=/Users/harry/mosflm625/bin/ipmosflm
export LOGFILE=mosflm625_osx_august_01.log
if [ ! -e $IPMOSFLM ]
then
    echo $IPMOSFLM doesn't exist - exiting now!!!
    exit
fi
#
echo Executable $IPMOSFLM | tee $LOGFILE
echo Running test on `date` >> $LOGFILE
#
I=1
while [ $I -le 10 ]
do
    TIME_USED=$( (time ${IPMOSFLM} < test_$I > mosflm.lp) 2>&1 > /dev/null )
    echo Run \#$I: cpu time: `echo $TIME_USED | awk '{print $4}'` >> $LOGFILE
    mv mosflm.lp mosflm_$I.lp
    mv lys_fine_002.mtz $I.mtz
    mv SUMMARY summary.$I
    /bin/rm -f GENFILE
    let I=I+1
done
#
I=1
while [ $I -le 10 ]
do
    wc -l mosflm_$I.lp IPMOSFLM_$I.lp
    let I=I+1
done
echo finished test at `date` >> $LOGFILE
```

```
[g4-15:~/test/lys_fine] harry% ./testit.sh
Executable /Users/harry/mosflm625/bin/ipmosflm
19010 mosflm_1.lp
19191 IPMOSFLM_1.lp
38201 total          (22313 lines different)
19366 mosflm_2.lp
19229 IPMOSFLM_2.lp
38595 total          23106      "      "
20329 mosflm_3.lp
19229 IPMOSFLM_3.lp
39558 total          25123      "      "
.
.
.
```

```
[g4-15:~/test/lys_fine] harry% more mosflm625_osx_august_01.log
Executable /Users/harry/mosflm625/bin/ipmosflm
Running test on Mon Aug 1 15:18:58 BST 2005
Run #1: cpu time: 0m46.886s
Run #2: cpu time: 0m50.305s
Run #3: cpu time: 1m0.763s
Run #4: cpu time: 0m58.744s
Run #5: cpu time: 0m56.463s
Run #6: cpu time: 0m54.628s
Run #7: cpu time: 0m51.343s
Run #8: cpu time: 1m2.693s
Run #9: cpu time: 0m47.770s
Run #10: cpu time: 0m56.151s
finished test at Mon Aug 1 15:29:01 BST 2005
```

```
! DO NOT ADD or REMOVE STUFF FROM THIS FILE
! It is intended to test mosflm in a background job with the
! following sequence of processes:
!
! (1) Autoindex from two images
! (2) estimate mosaicity from the first
! (3) postrefinement
! (4) integration
!
BEAM 149.79 150.87
GAIN 1.80
ADCOFFSET 6
DISTANCE 195.132
TEMPLATE lys_fine_###.pck
NEWMAT postref2.mat
MOSAIC ESTIMATE
AUTOINDEX DPS IMAGE 2 PHI 0 0.2 IMAGE 51 PHI 9.8 10.0
GO
POSTREF MULTI SEGMENT 2
PROCESS 2 TO 5 START 0 ANGLE 0.2
GO
PROCESS 47 TO 50 START 9 ANGLE 0.2
GO
#
POSTREF MULTI NOSE FIX ALL
PROCESS 2 TO 51 START 0 ANGLE 0.2
GO
```

```
[macf3c-3:~/test/lys_fine] harry% diff mosflm_2.lp IPMOSFLM_2.lp | wc -l
23106
[macf3c-3:~/test/lys_fine] harry% diff mosflm_2.lp IPMOSFLM_2.lp | more
1,3c1
<
<
< ***** Version 6.2.5 for Image plate and CCD data 9th August 2005
*****
---
> ***** Version 6.2.5 for Image plate and CCD data 30th June 2004
*****
.
.
.
511c487
< 149.74 150.90 1.0014 195.40 1.0002 14 20 0.070 -0.311 0.000 0.000
---
> 149.74 150.90 1.0014 195.40 1.0002 14 19 0.070 -0.311 0.000 0.000
651a628,631
>
> Detector distortion refinement using 50 SPOTS
> Starting residual=0.163mm; Weighted residual 0.87
> Residual after 1 CYCLE=0.115mm; Weighted residual 0.49
654c634
< 149.72 151.02 1.0010 195.33 0.9996 4 9 0.116 -0.322 0.000 0.000
---
> 149.72 151.02 1.0010 195.33 0.9996 4 8 0.116 -0.322 0.000 0.000
683a664,667
>
> Detector distortion refinement using 24 SPOTS
> Starting residual=0.090mm; Weighted residual 0.47
```

## Performance testing:

Is it an improvement over what existed before?

- produces results where it (or other software) didn't before
- faster (more streamlined code, better compilation, removal of bottlenecks)
- more accurate results (lower Rs, nicer peaks in maps)
- easier to use
- runs on a new platform

Can inform choice of hardware/OS/compiler/flags e.g. for the batch test series earlier:

	clock time
Linux, Pentium, 3.2GHz, g77 3.4, -O2:	8m 37s
Linux, Pentium, 1.5GHz, g77 3.2, -O1:	16m 51s
Linux, Pentium, 1.5GHz, ifc, -O3:	25m 53s
OS X, Mac, 1.67GHz, g77, -O0:	17m 33s
OS X, Mac, 1.67GHz, g77, -O2:	10m 22s
OS X, Mac, 1.67GHz, g77, -O5 -funroll-loops:	10m 09s
OS X, Mac, 2.0 GHz, XLF -O2:	4m 55s
Tru64, Alpha, 500MHz, f77 -O2:	5m 41s
Irix 6.5, SGI, 400MHz, f77 -O2:	24m 09s

Can highlight particular problems or indicate improvements:

e.g. for Linux, NFS mounted disks can cause severe performance problems - caused by local/remote handshaking every time a read or write is performed.

cure: (a) only use local disks  
(b) buffer i/o to reduce the number of transfers

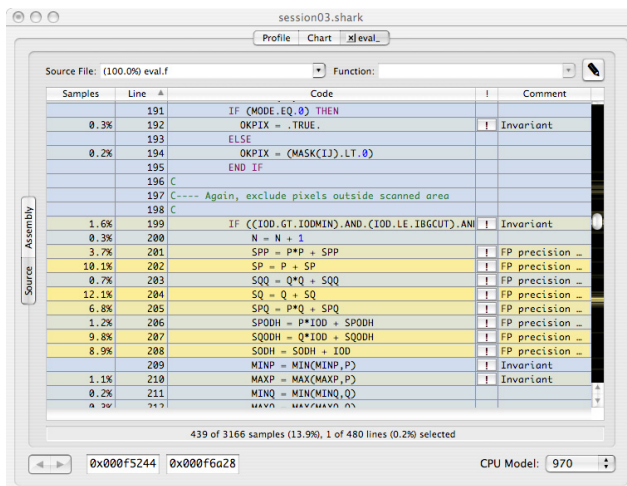
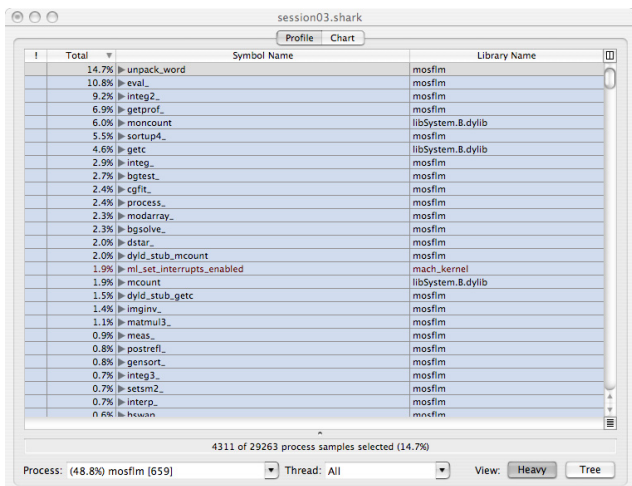
Profiling: use an external program to locate bottlenecks

e.g. Shark in OSX, gprof under other UNIXes; compile & link with flag "-pg", run the program and then

```
$ gprof <program> gmon.out
```

granularity: each sample hit covers 4 byte(s) for 0.03% of 39.25 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
12.4	4.87	4.87	115121	0.04	0.08	_eval_ [4]
11.7	9.48	4.61	43166	0.11	0.19	_integ2_ [6]
10.1	13.44	3.96	43166	0.09	0.09	_getprof_ [11]
8.5	16.76	3.32				_moncount_ (7093)
8.4	20.06	3.30	24	137.50	137.50	_rotate_clock90 [12]
6.5	22.61	2.55	115148	0.02	0.02	_sortup4_ [14]
5.7	24.84	2.23	24	92.92	92.92	_unpack_wordmar [16]
4.0	26.42	1.58	49069	0.03	0.11	_integ_ [10]
3.4	27.76	1.34	22	60.91	657.99	_process_ [3]
3.4	29.08	1.32				_mcount_ (152)



Finally - test at all stages of development.

Modern OOP methodology recommends producing test classes for all important methods to check they work with model data before inclusion into your main program. Having the test class available makes it easier to investigate when someone breaks your program with unexpected input.



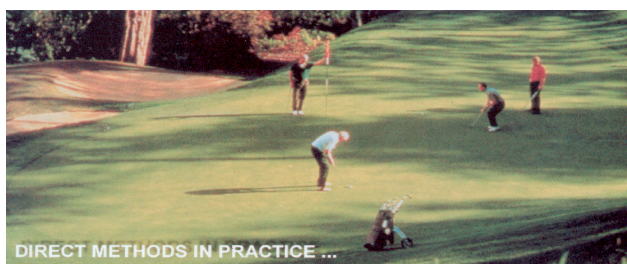
# The future of direct methods

IUCr Computing School, Siena,  
August 2005

George M. Sheldrick

<http://shelx.uni-ac.gwdg.de/SHELX/>

## Finding the minimum



## Normalized structure factors

Direct methods turn out to be more effective if we modify the observed structure factors to take out the effects of atomic thermal motion and the electron density distribution in an atom. The normalized structure factors  $E_h$  correspond to structure factors calculated for a point atom structure.

$$E_h^2 = (F_h^2/\varepsilon) / \langle F^2/\varepsilon \rangle_{\text{resl. shell}}$$

where  $\varepsilon$  is a statistical factor, usually unity except for special reflections (e.g. 00/ in a tetragonal space group).  $\langle F^2/\varepsilon \rangle$  may be used directly or may be fitted to an exponential function (Wilson plot).

## The crystallographic phase problem

- In order to calculate an electron density map, we require both the intensities  $I = |F|^2$  and the phases  $\phi$  of the reflections  $hkl$ .
- The information content of the phases is appreciably greater than that of the intensities.
- Unfortunately, it is almost impossible to measure the phases experimentally!

This is known as the **crystallographic phase problem** and would appear to be difficult to solve!

Despite this, for the vast majority of small-molecule structures the phase problem is solved routinely in a few seconds by black box **direct methods**.

## The Sayre equation

Sayre (1952). In the same issue of Acta Cryst., Cochran and Zachariasen independently derived phase relations and showed that they were consistent with Sayre's equation:

$$F_h = q \sum_{h'} (F_{h'} F_{h-h'})$$

where  $q$  is a constant dependent on  $\sin(\theta)/\lambda$  for the reflection  $h(hkl)$  and the summation is over all reflections  $h'$  ( $h'k'l'$ ). Sayre derived this equation by assuming equal point atoms. For such a structure the electron density ( $\rho$  or  $Z$ ) is proportional to its square ( $\rho^2$  or  $Z^2$ ) and the **convolution theorem** gives the above equation directly.

The Sayre equation is (subject to the above assumptions) exact, but requires complete data including  $F_{000}$ .

## The tangent formula (Karle & Hauptman, 1956)

The tangent formula, usually in heavily disguised form, is still a key formula in small-molecule direct methods:

$$\tan(\phi_h) = \frac{\sum_{h'} |E_{h'} E_{h-h'}| \sin(\phi_{h'} + \phi_{h-h'})}{\sum_{h'} |E_{h'} E_{h-h'}| \cos(\phi_{h'} + \phi_{h-h'})}$$

The sign of the sine summation gives the sign of  $\sin(\phi_h)$  and the sign of the cosine summation gives the sign of  $\cos(\phi_h)$ , so the resulting phase angle is in the range 0-360°.

## The Multan Era (1969-1986)

The program **MULTAN** (Woolfson, Main & Germain) used the tangent formula to extend and refine phases starting from a small number of reflections; phases were permuted to give a large number of starting sets. This **multisolution** (really multiple attempt) direct methods program was user friendly and relatively general, and for the first time made it possible for non-experts to solve structures with direct methods. It rapidly became the standard method of solving small-molecule structures.

Yao Jia-Xing (1981) found that it was even better to start from a large starting set with random phases (**RANTAN**), and this approach was adopted by most subsequent programs.

## Negative quartets: using the weak data too

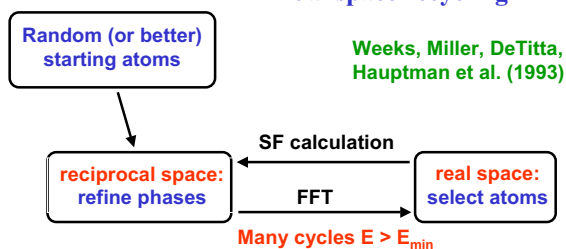
Schenk (1973) discovered that the quartet phase sum:

$$\Phi = \phi_h + \phi_{h'} + \phi_{h''} + \phi_{h-h'-h''}$$

is, in contrast to the TPR sum, more often close to  $180^\circ$  than  $0^\circ$  when the four primary  $E$ -values  $E_h$ ,  $E_{h'}$ ,  $E_{h''}$  and  $E_{h-h'-h''}$  are relatively large and the three independent cross-terms  $E_{h+h'}$ ,  $E_{h+h''}$  and  $E_{h'+h''}$  are all small. Hauptman (1975) and Giacovazzo (1976) derived probability formulas for these **negative quartets** using different approaches; Giacovazzo's formula is simpler and more accurate and so has come into general use.

Although this phase information is weak (and depends on  $1/N$  rather than  $1/N^{1/2}$  for TPRs) tests based on negative quartets discriminate well against uranium atom false solutions.

## Dual space recycling



If the figures of merit indicate a solution, it can be expanded to the complete structure using all data

Implemented in **SnB** and (later) **SHELXD**

## The correlation coefficient between $E_o$ and $E_c$

$$CC = \frac{100 [\sum (wE_o E_c) \sum w - \sum (wE_o) \sum (wE_c)]}{\{ [\sum (wE_o^2) \sum w - (\sum wE_o)^2] \cdot [\sum (wE_c^2) \sum w - (\sum wE_c)^2] \}^{1/2}}$$

Fujinaga & Read, *J. Appl. Cryst.* 20 (1987) 517-521.

For data to *atomic resolution*, a CC of 65% or more almost always indicates a correct solution.

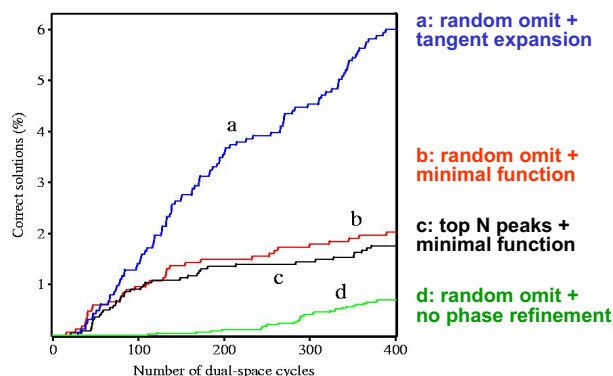
## Strategies for atom selection

- Simply keep top  $N$  atoms
- Eliminate atoms to maximize e.g.  $\sum E_c^2(E_o^2-1)$
- Eliminate 30% atoms at random

## Strategies for phase refinement

- Do no phase refinement
- Reduce the minimal function by the parameter-shift method
- Fix 30-50% of the phases with largest  $E_c$ , derive the rest by tangent expansion

## Gramicidin A (N=317) - different strategies



## Random OMIT maps

**Omit maps** are frequently used by protein crystallographers to reduce **model bias** when interpreting unclear regions of a structure. A small part (<10%) of the model is deleted, then the rest of the structure is refined (often with simulated annealing to reduce memory effects) and finally a new difference electron density map is calculated.

A key feature of SHELXD is the use of **random omit maps** in the search stage. About 30% of the peaks are omitted at random and the phases calculated from the rest are refined. The resulting phases and observed E-values are used to calculate the next map, followed by a peaksearch. This procedure is repeated 20 to 500 times.

Although the random omit and probabilistic Patterson sampling appreciably improve the efficiency of direct methods, using both together is not much better than either alone. Usually we use the probabilistic Patterson sampling for the location of heavy atoms for macromolecular phasing and random omit maps for *ab initio* structure solution.

## Unknown structures solved by SHELXD

Compound	Sp. Grp.	N(moi)	N(+solv)	HA	d(Å)
Hirustasin	P <sub>4</sub> <sub>3</sub> 2 <sub>1</sub> 2	402	467	10S	1.20
Cyclodextrin	P2 <sub>1</sub>	448	467		0.88
Decaplanin	P2 <sub>1</sub>	448	635	4Cl	1.00
Cyclodextrin	P1	483	562		1.00
Bucandin	C2	516	634	10S	1.05
Amylose-CA26	P1	624	771		1.10
Viscotoxin B2	P2 <sub>1</sub> 2 <sub>1</sub> 2 <sub>1</sub>	722	818	12S	1.05
Mersacidin	P3 <sub>2</sub> *	750	826	24S	1.04
Feglimycin	P6 <sub>5</sub> *	828	1026		1.10
Tsuchimycin	P1	1069	1283	24Ca	1.00
rc-WT Cv HiPIP	P2 <sub>1</sub> 2 <sub>1</sub> 2 <sub>1</sub>	1264	1599	8Fe	1.20
Cytochrome c3	P3 <sub>1</sub>	2024	2208	8Fe	1.20

\*twinned

## The 1.2 Å rule

"Experience with a large number of structures has led us to formulate the empirical rule that if fewer than half the number of theoretically measurable reflections in the range 1.1-1.2 Å are "observed", it is very unlikely that the structure can be solved by direct methods" [Sheldrick, 1990].

Morris & Bricogne, *Acta Cryst. D59* (2003) 615-617 gave an explanation: the variation of the experimental  $E^2$  with resolution shows that data in the range 1.2-1.0 Å have a higher **information content**.

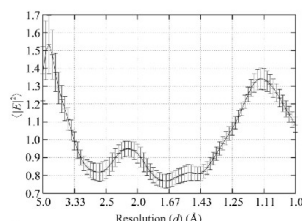


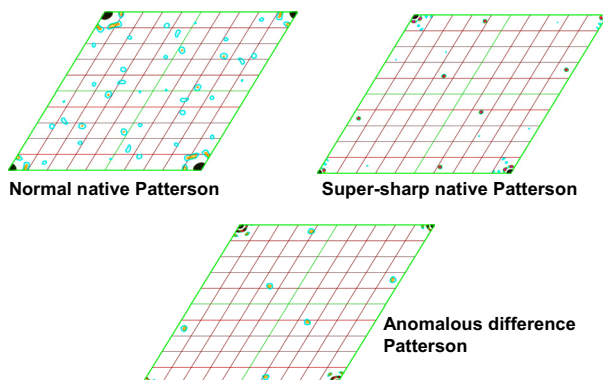
Figure 1  
Averaged squared normalized structure-factor amplitudes over 700 protein structures with standard deviations calculated from the population of individual  $|E|^2$  profiles.

## Heavy atoms and the 1.2 Å rule

When heavier atoms such as S or Fe are present, this rule can be relaxed a little. Tests using high resolution data artificially truncated (or not measured) to a resolution worse than the diffraction limit of the crystal also tend to perform better. Many of the largest structures solved by direct methods fall into these categories.

When heavy atoms are present, probabilistic sampling of a **super-sharp Patterson** [e.g. with coefficients  $\sqrt{E^2 F}$ ] is a good way to kick-start *ab initio* direct methods.

## Cytochrome c6 Pattersons



## Resolving the resolution problem

Replacing peak picking by some form of density modification, as used by Giacovazzo et al. in **SIR2003** and in **ACORN** (Yao Jia-Xing et al.) appears to alleviate the resolution problem a little, though maybe only to 1.3 or 1.4 Å. Recent versions of SIR (Giacovazzo et al.) make extensive use of iterative density modification, e.g. using only the strongest E-values and setting all but the highest density to zero. To judge from the published tests, **SIR2005** may be more effective than **SHELXD** and **SnB** for large structures at borderline atomic resolution, especially when heavier atoms are present.

A more far-reaching solution will probably be to find a clever way of exploiting chemical information that is not too expensive in terms of computer time. It should be noted that searching for small fragments instead of single atoms is particularly slow. We have successfully taken a small step in this direction by fitting  $S_2$ -units when locating the anomalous scatterers for sulfur-SAD phasing.



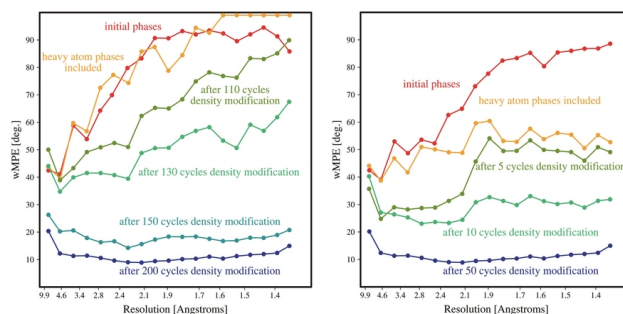
## Disulfide bond resolution

When the anomalous signal does not extend to sufficient resolution to resolve disulfides, it has been standard practice to search for *super-sulfur* atoms.

An effective alternative is to modify the peaksearch to locate the best positions for S-S units in the slightly elongated electron density maxima. These *resolved disulfides* not only improve the performance of the substructure solution, they also give a much better phase extension to higher resolution and better final map correlation coefficients. The CPU time overhead is negligible.

This suggests that searching for small fragments in the real space part of the dual-space recycling may be a good way of extending direct methods to lower resolution, provided that it can be done efficiently.

## Mean phase errors as a function of resolution for SAD phasing of cubic insulin at a wavelength of 1Å



without disulfide resolution

with disulfide resolution

## Other promising methods for atomic resolution data

1. **Iterated projections** [Elser, *Acta Cryst.* A59 (2003) 201-209]. This is a complicated iterative density modification algorithm that seems to be quite effective and not much slower than SnB or SHELXD. So far it is restricted to space group P1.
2. **Charge flipping** [Oszlanyi & Suto, *Acta Cryst.* A60 (2004) 134-141; A61 (2005) 147-152; Wu et al., A60 (2004) 326-330]. This algorithm is simple and easy to program, but in my tests was not quite as effective as the almost as simple *random omit method* (on its own without the tangent formula etc.).
3. **Integer programming** [Viai & Sahinidis, *Acta Cryst.* A59 (2003) 452-458 & A61 (2005) 445-452]. By reducing **CENTROSYMMETRIC** direct methods to an *integer programming problem*, it appears that these authors have indeed found a solution to the phase problem *in polynomial time*. Tests have shown that this method is as fast or faster, and less likely to fail, than existing methods for centrosymmetric structures.

## Ab initio direct methods at lower resolution

Considerable progress has also been made at very low resolution, where the number of reflections is so small that it is feasible to test many phase permutations [e.g. Lunina, Lunin & Urzhumtsev, *Acta Cryst.* D59 (2003) 1702-1713]. Solutions are selected on the basis of good connectivity and the right number of connected fragments in the cell, and then merged with each other. In favourable cases it may be possible to begin to see secondary structure in the 4 to 6 Å maps that are produced. Such methods are very sensitive to missing or wrongly measured low order reflections.

**Conditional optimisation** [Scheres & Gros, *Acta Cryst.* D57 (2001) 1820-1828] is a sort of molecular dynamics with N atoms in a box subject to a very general force field so that chemically sensible ensembles of atoms are favoured. In principle this is a promising method, but will probably require massive computer resources.

## Structure solution in P1

It has been observed [e.g. Sheldrick & Gould, *Acta Cryst.* B51 (1995) 423-431; Xu et al., *Acta Cryst.* D56 (2000) 238-240; Burla et al., *J. Appl. Cryst.* 33 (2000) 307-311] that it may be more efficient to solve structures in P1 and then search for the symmetry elements later. This works particularly well for solving P1 structures in P1.

I thought that this might be a good way of tackling problems where the space group is not clear. A decision as to the space group could simply be postponed until the structure has been solved! However after much effort I have come to the conclusion that, although the approach works well in straightforward cases, in pseudosymmetry cases there may be a problem in recognising the correct solution to the phase problem, so the current procedure of trying all possible space groups may be more effective!

More than 90% of the algorithms I have devised and programmed turned out, on objective assessment, not to represent improvements on current practice. This was simply one more example.

## Acknowledgements

I am particularly grateful to Isabel Usón, Thomas R. Schneider, Stephan Rühl and Tim Grüne for many discussions.

**SHELXD**: Usón & Sheldrick (1999), *Curr. Opin. Struct. Biol.* 9, 643-648; Sheldrick, Hauptman, Weeks, Miller & Usón (2001), *International Tables for Crystallography Vol. F*, eds. Arnold & Rossmann, pp. 333-351; Schneider & Sheldrick (2002), *Acta Cryst.* D58, 1772-1779.

**SHELXE**: Sheldrick (2002), *Z. Kristallogr.* 217, 644-650; Debreczeni, Bunkóczi, Girmann & Sheldrick (2003), *Acta Cryst.* D59, 393-395; Debreczeni, Bunkóczi, Ma, Blaser & Sheldrick (2003), *Acta Cryst.* D59, 688-696; Debreczeni, Girmann, Zeeck, Krätznér & Sheldrick (2003), *Acta Cryst.* D59, 2125-2132.

<http://shelx.uni-ac.gwdg.de/SHELX/>



## Clipper

Using the Clipper libraries.

Kevin Cowtan

[cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Why use a crystallographic library?

- Because it will save you a huge amount of work:
  - 3-10x increase in productivity.
  - Common algorithms are already built-in.
  - Well designed classes prevent common coding errors.
- Because it has been extensively debugged.
  - by use in other programs.
  - by test suites of varying degrees of formality.

Why not?

- Because its more to learn, and more to build.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Why use a Clipper in particular?

- Purpose designed for phase improvement and interpretation, i.e. good for:
  - Phasing
  - Phase improvement
  - Refinement
  - Model building

Why not?

- No facilities for un-merged data.
- Limited facilities for anything before fixing origin.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Why use a Clipper in particular?

- Mainly for C++ use.
- For Fortran or scripting purposes, write a wrapper which does most of the task and communicates in an appropriate way with the problem concerned.

Why not?

- No general scripting interface.
  - (Although it certainly works with SWIG.)

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

How to get Clipper:

- From my website:
  - <http://www.ysbl.york.ac.uk/~cowtan/clipper/clipper.html>
  - Simple script based build system.
- With CCP4
  - GNU autoconf build system.
- With CCTBX
  - SCONS build system.

I'm not an expert on build systems!

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

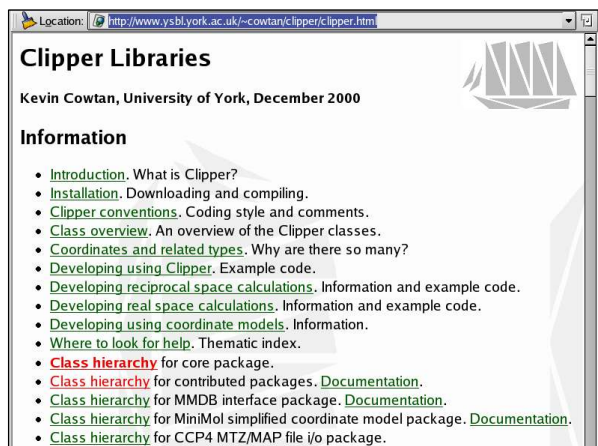
How to get information on Clipper:

- From my website:
  - <http://www.ysbl.york.ac.uk/~cowtan/clipper/clipper.html>
  - Comprehensive 'doxygen' documentation, including:
    - class codumentation
    - a range of tutorials
    - a few examples: more are in the 'examples' directory.
  - Aside: When you write code:
    - document it.
    - write test code.
      - (Proper test classes are better than stand-alone test programs – I'm working on it right now).

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries



Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Building Clipper applications:

- When compiling, `-I$INCLUDEDIR`
  - Usually `$PREFIX/include`
- When linking, `-L$LIBDIR`
  - Usually `$PREFIX/include`
- Using my build scripts, to make a simple program put a single `.cpp` file in the `examples` directory, and use  
`make program_name`

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

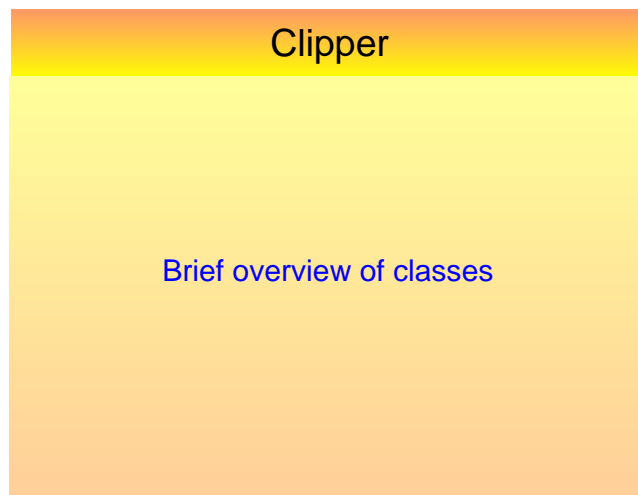
## Clipper libraries

Clipper conventions:

- Main classes in '`clipper`' namespace.
- Data may be held at either precision: data classes in '`clipper::data32`', '`clipper::data64`'
- Small objects use 'double' by default, but 'float' is also possible.
- Terminology:
  - fractionals are (u,v,w).
  - $4\sin^2 / \lambda^2$  is 'invresolsq'

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper



Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Overview of classes:

- Helper classes
  - `Clipper::String`
    - An interchangeable extension to `std::string`, with additional parsing and manipulation methods (e.g. `split`, `strip`).
  - `Clipper::Util`
    - A collection of useful static functions providing common crystallographic functions (e.g.  $I_i/I_0$ , B-U conversion) and portability code.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Overview of classes:

- Crystal information
  - cell and symmetry
- Ordinates, derivatives, operators and grids
  - HKLs, coordinates etc.
- Data classes
  - reflection data, maps (crystallographic and otherwise)
- Method objects
  - common tasks, e.g. data conversion, `sigmaa`, etc.
- Input/output classes

## Clipper libraries

### Crystal information:

- A crystal is defined by two main classes: a unit cell (`clipper::Cell`) and a spacegroup (`clipper::Spacegroup`).
- These are complex classes which store derived information and provide optimised methods for handling it.
- Two smaller 'descriptor' objects provide a more compact representation for storage and transmission: The cell descriptor (`clipper::Cell_descr`) holds just the cell edges and angles, and the spacegroup descriptor (`clipper::Spgr_descr`) holds the 'signature' of the spacegroup.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

### Ordinates and grids:

- Ordinates include Miller indices (`clipper::HKL`), orthogonal and fractional coordinates (`clipper::Coord_orth`, `clipper::Coord_frac`), grid coordinates (`clipper::Coord_grid`), and others.
  - The ordinates have methods to convert to any other related form.
- Operators for transforming coordinates. Rotation matrices and rotation translation operators (e.g. `clipper::RTop_orth`, `clipper::RTop_frac`, `clipper::Symop`).
  - Operators also contain transformation methods.
- Gradients, curvatures, grids (`clipper::Grid_sampling`), etc.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

### Data objects:

- Data objects hold the actual crystallographic data. They include reciprocal space data (`clipper::HKL_info`, `clipper::HKL_data`), crystallographic and non-crystallographic maps (`clipper::Xmap`, `clipper::NXmap`), and FFT maps (`clipper::FFTmap`).
- The primary design goal of the data objects is that they hide all the bookkeeping associated with crystallographic symmetry (and in real space, cell repeat). Data can be written to and read from any region of real or reciprocal space, and the unique stored copy of the data will be modified correctly. This is all achieved in a computationally efficient manner.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

### Method Objects:

- For common crystallographic tasks:
  - calculating functions of resolution.
  - calculating structure factors from coordinates.
  - sigma-a, likelihood maps.
  - map filtering.
  - map alignment, feature recognition, skeletonisation.
- Constructor creates a (tiny) method object setting any parameters for the calculation.
- `operator(...)` method does the actual calculation.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

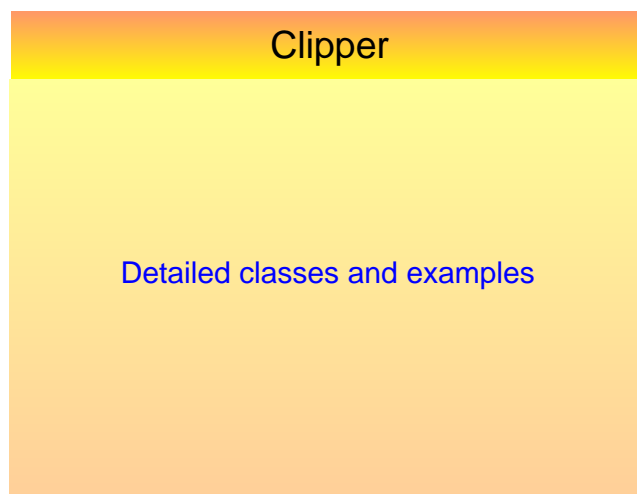
## Clipper libraries

### Input/Output objects

- Input/Output objects are used to record the contents of an object in a file or restore the contents from a file.
- Different objects are used for different file types, but the interfaces are as similar as the file format allows.
- (`CCP4MTZfile`, `CCP4MAPfile`, `MMCIFfile`, `CNSfile`, `PHSfile`)

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper



## Clipper Libraries: In detail

### Crystal Information: Cell and Spacegroup

- We almost never make objects from scratch, but we can using the compact 'descriptions':  
`Cell_descr celld(30.0,40.0,50.0);`  
`Cell cell( celld );`  
  
`Spgr_descr spgrd( "P 2ac 2ab" );`  
`Spacegroup spgr( spgrd );`
- Cell description takes up to 6 arguments (a,b,c,alpha,beta,gamma), degrees or radians.
- Spacegroup description can be H-M or Hall symbol, spacegroup number, or list of operators.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper Libraries: In detail

### Crystal Information: Cell

- We can access cell properties:  
  
`double a = cell.a();`  
`double astar = cell.a_star();`  
`double vol = cell.volume();`  
`Mat33<> mat = cell.matrix_orth();`
- Many clipper objects have an 'null' state, which they are in if not initialised, and an 'init' method:  
  
`if ( cell.is_null() )`  
`cell.init( clipper::Cell_descr( ... ) );`

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper Libraries: In detail

### Crystal Information: Spacegroup

- We can access symmetry operators:  
`int nsym = spgr.num_symops();`  
`int nsymp = spgr.num_primitive_symops();`  
`Symop op = spgr.symop[i];`
- Asymmetric units:  
`if ( spgr.in_asu( hkl ) )`  
`...`
- Reflection centric/absence/multiplicity:  
`HKL_class cls = spgr.hkl_class( hkl );`
- And much more.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper Libraries: In detail

### Coordinates:

- Can construct from 3 numbers, or `Vec3<>`.
- Can convert orthogonal <-> fractional using `Cell`.
- Can convert fractional <-> grid using `Grid_sampling`.
- Can transform using `coord.transform(rtop)` or `rtop*coord`

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

<code>Coord_orth(const ftype &amp;x, const ftype &amp;y, const ftype &amp;z)</code> <i>constructor: from x,y,z</i>
<code>Coord_orth(const Coord_orth &amp;x1, const Coord_orth &amp;x2, const Coord_orth &amp;x3, const ftype &amp;length, const ftype &amp;angle, const ftype &amp;torsion)</code> <i>constructor: from 3 coords and bond length, angle, torsion</i>
<code>const ftype &amp; x() const</code> <i>get x</i>
<code>const ftype &amp; y() const</code> <i>get y</i>
<code>const ftype &amp; z() const</code> <i>get z</i>
<code>ftype lengthsq() const</code> <i>return square of length of vector in Angstroms</i>
<code>Coord_frac coord_frac(const Cell &amp;cell) const</code> <i>orthogonal-fractional coordinate conversion</i>
<code>Coord_orth transform(const RTop_orth &amp;op) const</code> <i>return transformed coordinate</i>
<code>String format() const</code> <i>return formatted String representation</i>

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper Libraries: In detail

### Coordinates:

- Example transformations...  
`// Make grid coord from ints`  
`Coord_grid cg( u, v, w );`  
`// convert to fractional using grid`  
`Coord_frac cf = cg.coord_frac( grid );`  
`// transform using symop`  
`cf = spgr.symop(2) * cf;`  
`// convert to orthogonal using cell`  
`Coord_orth co = cf.coord_orth( cell );`  
`// format and print the result`  
`std::cout << co.format() << "\n";`

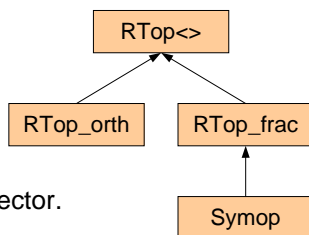
Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper Libraries: In detail

Operators:

- Rotation-translation operator most common, consists of rotation matrix and translation vector.
- Construct from matrix and vector.
- Can apply to any coordinate in the same system.
- Can combine multiple RTop-s by multiplication.



Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Clipper

## Clipper Libraries: In detail

Operators:

- Rotations may also be represented by Euler angles (CCP4 or full 24 conventions), polar angles (CCP4 convention), or quaternion (interchange format).
 

```

//Make euler rotation (radians)
Euler_ccp4 euler( pi/3, pi/4, pi/5 );
// convert to quaternion
Rotation rot( euler );
// convert to polar
Polar_ccp4 polar = rot.polar_ccp4();
// make vector
Coord_orth co( 1.0, 2.0, 3.0 )
// make RTop_orth
Rtop_orth op( rot.matrix(), co );
      
```

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Clipper

## Clipper Libraries: In detail

HKLs:

- Construct from integers, access h, k, l:
 

```

HKL hkl(1,2,3);
int h = hkl.h();
      
```
- Calculate resolution:
 

```

double s = hkl.invresolsq( cell );
      
```
- Transform:
 

```

HKL equiv1 = spgr.symop(i) * hkl;
HKL equiv2 = spgr.isymop(i) * hkl;
      
```
- Calculate phase shift, etc:
 

```

double dphi = hkl.sym_phase_shift( symop );
      
```

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Clipper

## Clipper libraries

Data objects:

- Reciprocal space:
  - We often want to handle many lists of related reflection data, handling all the data connected with one HKL at once.
  - We often want to add new data during the course of the calculation.
  - Some data are tied together.
- Clipper implements a system of data lists, holding data of crystallographic types, using a common indexing defined by a parent object.

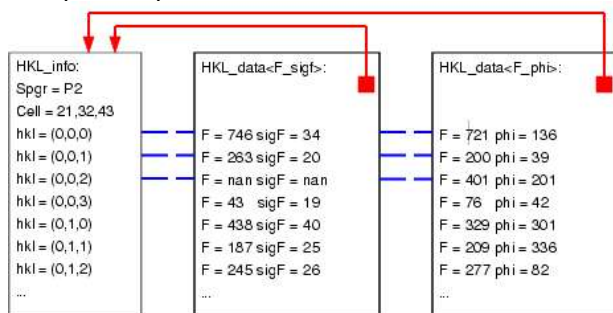
Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Clipper

## Clipper libraries

Data objects:

- Reciprocal space:



Note: Data types may be complex, e.g. F+/F-, ABCD

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Clipper

## Clipper libraries

Data objects:

- Reciprocal space:
  - `clipper::HKL_info` manages the list of HKLs for all the objects.
  - `clipper::HKL_data<type>` holds a single list of data of some crystallographic type. (e.g. F/sigF, A/B/C/D, I(+)/I(-)/sigI(+)/sigI(-) )
- Note this is the scheme in Clipper version 1. In version 2 (coming soon), `clipper::HKL_info` objects will disappear, being created and destroyed in the background automatically as required. Existing code is not affected!

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Clipper

## Clipper libraries

Data objects:

- Reciprocal space:

```
// make crystal objects
Spacegroup spgr( ... );
Cell cell( ... );
Resolution reso( 3.0 );
// make reflection list
HKL_info hklinf( spgr, cell, reso, true );
// make data lists using reflection list
HKL_data<data32::F_sigF> fsig( hklinf );
HKL_data<data32::Phi_fom> phiw( hklinf );
HKL_data<data32::ABCD> abcd( hklinf );
```

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Data objects:

- Reciprocal space: Can access data by index in list, or by HKL. If the HKL requested isn't in the list, the correct symmetry equivalent will be chosen, and the data transformed automatically (including Friedel and phase shift).

```
// get 23rd data by index
double f = fsig[23].f();
double sig = fsig[23].sigf();
// get (1,2,3) data, (-1,-2,-3) data
double phi1 = fphi[ HKL(1,2,3) ];
double phi2 = fphi[ HKL(-1,-2,-3) ];
// by definition, phi1 = -phi2
```

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Data objects:

- Data types are complex objects, from which individual items may be accessed. They know how to transform themselves about reciprocal space. The user can define new data types.
- Operators (+, -, \*, &, |, !) are available to add, subtract, or scale entire lists when these make sense.
- 'Compute operators' are available for applying common operations and conversions to data, e.g.:
  - ABCD <-> phi/fom
  - Scaling by scale, overall B
  - F -> semi-normalised E

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Data objects:

- Conversions are available for transforming between types, individually or as a list. (Lambda operators).

```
HKL_data<data32::F_sigF> fsig( hklinf );
HKL_data<data32::ABCD> abcd( hklinf );
HKL_data<data32::Phi_fom> phiw( hklinf );
HKL_data<data32::F_phi> fphi( hklinf );
// convert abcd to phi/fom
phiw.compute( abcd, Compute_phifom_from_abcd() );
// convert F/sigF and phi/fom to weighted F/phi
fphi.compute( fsig, phiw,
               Compute_fphi_from_fsifg_phifom() );
```

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Data objects:

- These methods provide convenient access when performance isn't an issue. But access by HKL involves symmetry search, so can be slow. When performance is critical, an alternative approach is adopted, using 'reference' objects, which are slightly related to STL iterators. These are designed to optimise common access patterns:
- When looping over all data sequentially:  
`HKL_data<*>::HKL_reference_index`
- When accessing data by HKL rather than by index:  
`HKL_data<*>::HKL_reference_coord`

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Data objects:

- The index is like a normal array index, but can also return resolution and other information. It may be used for any `HKL_data` with the same `HKL_info`.  
`HKL_data<data32::F_phi>::HKL_reference_index ih;`  
`for ( ih = fphi.first(); !ih.last(); ih.next() ) {`  
 `HKL hkl = ih.hkl();`  
 `double s = ih.invresolsq();`  
 `data32::F_phi fp = fphi[ih];`  
`}`
- The coordinate reference type allows fast access to neighbouring and nearby reflections (for which the symmetry operator for the stored ASU is usually conserved).

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Data objects:

- Indices and references may be used across any lists which share the same HKL\_info.
- To transfer data between differently indexed lists, you must go back to the underlying HKL.
  - Loop over the target HKL\_data and fetch data from the source HKL\_data by HKL.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Data objects:

- Crystallographic and non-crystallographic maps (`clipper::Xmap`, `clipper::NXmap`)
  - The data objects are templates which can hold data of any type. In the case of a map, this type will usually be 'double' or 'float'.
  - Xmap-s have crystallographic symmetry and lattice repeat.
  - Nxmap-s have neither, and define a bounded region in the coordinate space.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Data objects:

- Crystallographic maps (`clipper::Xmap`)
  - Construct from `Spacegroup`, `Cell`, `Grid_sampling`.
  - Usually construct `Grid_sampling` from `Spacegroup`, `Cell`, `Resolution`.

```
Grid_sampling grid( spgr, cell, reso );
Xmap<float> xmap( spgr, cell, grid );
```
  - Can the calculate maps by FFT from `HKL_data<F_phi>`

```
xmap.fft_from( fphi );
xmap.fft_to( fphi );
```

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Data objects:

- Crystallographic maps (`clipper::Xmap`)
  - Access map by index, or coordinate:

```
// allowed but discouraged
float rho1 = xmap.get_data(1234);
float rho2 = xmap.get_data(Coord_grid(1,2,3));
```
  - Fine for some tasks. However:
    - Indices are not sequential, owing to irregular shape of ASU.
    - Access by coordinate requires symmetry lookup to find value in stored ASU.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Data objects:

- Crystallographic maps (`clipper::Xmap`)
  - Access map by two reference types when performance is important:

```
// preferred: reference index
Xmap::Map_reference_index ix;
for ( ix = xmap.first(); !ix.last(); ix.next() )
    float rhox = xmap[ix];

// preferred: reference coord
Xmap::Map_reference_coord iy( xmap, Coord_grid(1,2,3) );
float rho1 = xmap[iy];
iy.next_u();
iy.prev_w();
float rho2 = xmap[iy];
// now iy -> Coord_grid(2,2,2)
```
  - References may be shared across similar maps.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Data objects:

- Crystallographic maps (`clipper::Xmap`)
  - Methods are provided for interpolation, and also gradient and curvature calculation:

```
// get interpolated density
Coord_frac cf( 0.1, 0.2, 0.3 );
float rho1 = xmap.interp<Interp_linear>( cf );
float rho2 = xmap.interp<Interp_cubic>( cf );
```
  - Also sorting, statistics, etc.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper



## Clipper libraries

### Data objects:

- Indices and references may be used across any maps which share the same space-group and grid (similar to reflection data).
- To transfer data between differently indexed maps, you must go back to the underlying `Coord_grid`, or `Coord_orth` if cells are different (interpolate).
  - Loop over the target `Xmap` and fetch data from the source `Xmap` by `Coord_grid`.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

### Data objects:

- Atoms:  
(`clipper::Atom`, `clipper::Atom_list`)
  - `clipper::Atom`: The simplest definition necessary for electron density calculation.
  - `clipper::Atom_list`: Derived from `std::vector<Atom>`.
  - For more complex atom manipulation, see the MMDB interface and the MiniMol package.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

### Method objects:

- There are a range of common calculations provided in to the Clipper packages:
  - Resolution functions – used to calculate smoothly varying estimates of things in reciprocal space, e.g.  $|F(h)|$  for normalisation of E's.
  - Electron density/structure factor calculation from atoms.
  - Map filtering, e.g. for calculating local mean of variance of electron density.
  - Skeletonisation.
  - Likelihood weighting and map calculation.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

### Method objects:

- There are a range of common calculations provided in to the Clipper packages:
  - Electron density/structure factor calculation from atoms.  
`SFcalc_iso_fft sfcalc;`  
`Atom_list atoms;`  
`...`  
`sfcalc( fphi, atoms );`
  - Constructor for `SFcalc_iso_fft` can take optional arguments to control it's behaviour.
  - The actual calculation is done by the `()` operator. The result is the first argument.
    - Note: there are several implementations (e.g. slow/fft/iso/aniso).

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

### Method objects:

- Other examples:
  - Map filtering: first we define the filter function to apply to the map, and then we apply it using a given filter implementation.  
`MapFilterFn_step fn( filter_radius );`  
`MapFilter_fft<float> fltr( fn, 1.0, Relative );`  
`Xmap<float> filtered;`  
`fltr( filtered, xmap );`
  - This filters with a step function of a given radius, using the scaling parameters 1.0 and 'Relative', and puts the result in the new map.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

### Method objects:

- Other examples:
  - Resolution functions are more complex:
    - First, we define the curve we want to fit.
    - Secondly, we define the function the curve must minimise.
    - Thirdly, we define a set of initial parameters.
    - Fourthly, we feed both of these to an evaluator.
  - e.g. fitting a spline function to scale two datasets together:  
`BasisFn_spline basisfn( 6 );`  
`TargetFn_scaleF1F2<data32::F_sigF,data32::F_sigF>`  
`targetfn( fsig1, fsig2 );`  
`std::vector<double> params( 6, 1.0 );`  
`clipper::ResolutionFn`  
`rfn( hklinf, basisfn, targetfn, params );`

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Method objects:

- Other examples:

```
- e.g. fitting a spline function to scale two datasets together:
// fitting the function
BasisFn_spline basisfn( 6 );
TargetFn_scaleF1F2<data32::F_sigF,data32::F_sigF>
    targetfn( fsig1, fsig2 );
std::vector<double> params( 6, 1.0 );
clipper::ResolutionFn
    rfn( hklinf, basisfn, targetfn, params );

// scaling the data
for ( ih = fsig1.first(); !ih.last(); ih.next() )
    fsig1[ih].scale( sqrt( rfn(ih) ) );
```

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Clipper

## Clipper libraries

Input/output objects:

- Open an I/O object for a particular file type, for read or write, and then import/export the data object concerned using that object.
  - e.g. For a crystallographic map:

```
Xmap<float> map;
CCP4MAPfile mapfile;
mapfile.open_read( "1ajr.map" );
mapfile.import_xmap( map );
mapfile.close();
```
  - For export, `open_write()` and then `export_xmap()`.
  - Can also import/export `NXmap`-s from the same object.

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Clipper

## Clipper libraries

Input/output objects:

- Reflection files vary. PHS files are simple. CNS and mmCIF files intermediate. MTZ files contain multiple sets of data of various types, and accompanying information.
  - e.g. Import a set of F-s and  $\sigma$ -s:

```
CCP4MTZfile mtzfile;
mtzfile.open_read( "1ajr.map" );
mtzfile.import_hkl_info( hklinf );
mtzfile.import_hkl_data( fsig, "/*/[/FP,SIGFP]" );
mtzfile.close();
```
  - Can also import/export spacegroup, cell, and dataset information, where the format supports it.

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Clipper

## Clipper libraries

Trivial examples:

- Read some data, calculate a map.
  - Assume F and phi.  
(We've seen how to get these from phi/fom, ABCD.)
- Expand data to P1
  - Including all appropriate symmetry transformations.
- Rotating density from an Xmap into an Nxmap

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Clipper

## Clipper libraries

Calculate a map from structure factors:

```
clipper::HKL_info hkl; // define hkl objects
clipper::HKL_data<clipper::data32::F_phi> fphidata( hkl );
// READ DATA
clipper::CCP4MTZfile mtzin;
mtzin.open_read( "my.mtz" ); // open new file
mtzin.import_hkl_info( hkl ); // read sg, cell, reso, hkl
mtzin.import_hkl_data( fphidata, "/*/[/FCAL,PHICAL]" );
mtzin.close_read();
// DEFINE MAP
clipper::Grid_sampling mygrid( hkl.spacegroup(), hkl.cell(),
                               hkl.resolution() ); // grid
clipper::Xmap<float> mymap( hkl.spacegroup(), hkl.cell(),
                           mygrid ); // map
mymap.fft_from( fphidata ); // fill map
// OUTPUT MAP
clipper::CCP4MAPfile mapout;
mapout.open_write( "my.map" ); // write map
mapout.export_xmap( mymap );
mapout.close_write();
```

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Clipper

## Clipper libraries

Expanding data to lower symmetry:

```
// make new objects
clipper::HKL_info newhkl( newspacegroup,
                           hkl.cell(), hkl.resolution() );
clipper::HKL_data<clipper::data32::F_phi> newfphidata(newhkl);
// and fill them
HKL_info::HKL_reference_index ih;
for ( ih = newhkl.first(); !ih.last(); ih.next() )
    newfphidata[ih] = fphidata[ih.hkl()];

// same thing to expand a map
// make new object
newmap.init( newspacegroup, map.cell(), map.grid_sampling() );
// and fill it
clipper::Xmap_base::Map_reference_index ix;
for ( ix = newmap.first(); !ix.last(); ix.next() )
    newmap[ ix ] = map.get_data( ix.coord() );
```

Kevin Cowtan, cowtan@ysbl.york.ac.uk

Sienna/Clipper

## Clipper libraries

Expanding data to lower symmetry:

```
// make new objects
clipper::HKL_info newhkl( newspacegroup,
                          hkl.cell(), hkl.resolution() );
clipper::HKL_data<clipper::data32:f_phi> newfphidata(newhkl);
// and fill them
HKL_info::HKL_reference_index ih;
for ( ih = newhkl.first(); !ih.last(); ih.next() )
    newfphidata[ih] = fphidata[ih.hkl()];
```

Important rule of thumb for reflections and maps:

- When moving data into an object with organization, always loop through the target object, and set each element by fetching data from the source object by coordinate.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Rotating a map:

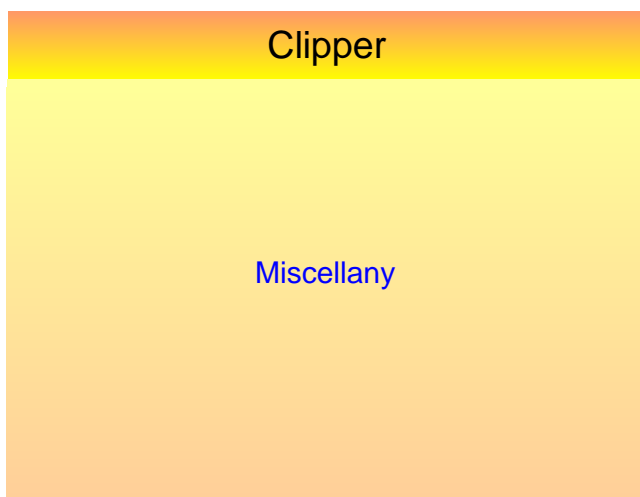
```
// make objects
clipper::Xmap<float> xmap;
clipper::NXmap<float> nxmap;
clipper::RTop_orth rtop

// initialise objects
...

// do the rotation
clipper::NXmap_base::Map_reference_index ix;
for ( ix = nxmap.first(); !ix.last(); ix.next() )
    nxmap[ix] = xmap.interp<Interp_cubic>( rtop*ix.coord_orth() );
```

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper



Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

CCP4 Extras:

- A mini-package of tools for building CCP4 command line programs:
  - A parser class, which combines input from both command line and standard input.
  - A CCP4-program class, which calls all the normal functions at the beginning of a program, and tidies up at the end.
    - Note: don't use exit. Continue to the end of a block.
- Currently just a source file which you can link to your program.
  - Possible part of libclipper-ccp4 in future?

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Learning more:

- Documentation contains several tutorials, plus extensive reference material (~1000 pages)
- Lots of material in the 'examples' directory:
  - map calculation
  - structure factor calculation
  - ML weighting and maps
  - data analysis
  - data conversion
  - scaling
  - etc.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Clipper libraries

Conclusions:

- Clipper contains a great many building blocks for the rapid construction of crystallographic calculations.
- It is suitable for a certain range of problems, based on language and stage of structure solution.
- Convenience methods provide very simple ways to do complex tasks.
- Optimised methods provide near-optimal performance in terms of CPU and memory usage.
- Lots of documentation and examples...

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Clipper

## Map manipulation

### Exercises in map manipulation.

Kevin Cowtan  
[cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Map manipulation

## Map manipulation

### Exercises:

(Attempt any or all of these in any order. Don't expect to finish them during the meeting)

- Identify connected regions in a mask
- Skeletonise an electron density map
- Calculate electron density from atomic coordinates
  - Each task can be more or less difficult depending on whether you account for lattice repeat, space-group symmetry, and cell geometry.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Map manipulation

## Map manipulation

### Identify connected regions in a mask

- The file mask.txt contains a 12x10x8 mask, in obvious human readable format. Read this into an array, and using an algorithm of your choice, identify how many discrete connected masked regions there are. Mark each connected region in the mask with a unique identifier. Print a list of them, along with their sizes.
- For a more advanced solution, include the fact that the map is cyclic and wraps round at its edges.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Map manipulation

## Map manipulation

### Skeletonise an electron density map

- The file map.txt contains a 12x10x8 map, in obvious human readable format. Read this into an array, and calculate a modified Greer skeleton, using the algorithm described.
- For a more advanced solution, include the fact that the map is cyclic and wraps round at its edges.  
If writing this using Clipper, you should also be able to incorporate crystallographic symmetry. (If you store the skeleton in an Xmap, then it will also have magic symmetry. This is how 'coot' displays infinite skeletons.)
- Are there any changes you would make to the program to support skewed grids?

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Map manipulation

## Map manipulation

### Calculate electron density from atomic coordinates

- Using a blank 12x10x8 map representing a 18x15x9A cell, read the coordinates from the file atoms.txt. Calculate the electron density for the unit cell, assuming that each atom is a Gaussian whose height is its atomic number and whose half width is 0.75A.
- For a more advanced solution, include the fact that the map is cyclic and wraps round at its edges.
- How must the calculation be modified to handle skewed cells? What types of coordinates are involved?
- If you are using Clipper, the file clipper/contrib/edcalc.cpp contains a symmetry general solution: `Edcalc_iso<T>::operator()`. Why do you think a multiplicity correction is required at the end of the calculation?

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Map manipulation

## Map manipulation

### Skeletonisation:

- Aim is to trace the ridges connecting peaks of density in the map.
- Simplest approach is a modified 'Greer' algorithm.
  - This version descended from one implemented in 'dm' in the late '90s.
  - Also similar to one used in TEXTAL.
- A more general version (symmetry and crystal geometry) is implemented in Clipper.

Kevin Cowtan, [cowtan@ysbl.york.ac.uk](mailto:cowtan@ysbl.york.ac.uk)

Sienna/Map manipulation

# Map manipulation

Skeletonisation:

- Make a map of flags, using the same grid as the density map. Mark every point as 'skeleton'.
- Consider each grid point in the map in turn, in order of increasing density.
  - For each point, consider whether removing that point from the skeleton will 'disconnect' any of its (6 orthogonally adjacent) neighbours.
    - If so, leave it in the skeleton.
    - If not, remove it from the skeleton.

# Map manipulation

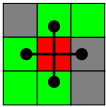
Skeletonisation:

- e.g. in 2 dimensions...

In skeleton

Not in skeleton

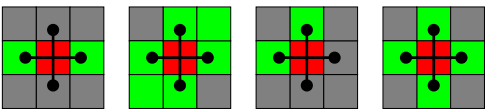
Testing
- So in this example, removing the center point will disconnect the top neighbor from the left and bottom neighbors.
  - So we keep it.



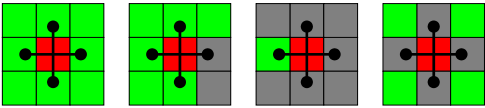
# Map manipulation

Skeletonisation:

- Keep:



- Lose:



# Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26

# Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26

# Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26

Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26

Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26

Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26

Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26

Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26

Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26

Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26

Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26

Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26

Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26

Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26

Map manipulation

Skeletonisation:

25	3	8	24	17	18
21	7	29	14	11	28
10	30	23	13	6	20
4	15	16	9	2	12
22	19	27	5	1	26



# Reciprocal Space Tutorial

IUCr Computing School, Siena,  
August 2005

George M. Sheldrick

<http://shelx.uni-ac.gwdg.de/SHELX/>

## Fortran-90 example

The following Fortran-90 program is provided as a solution to Kevin's first exercise [finding separate domains in a cyclic mask consisting of 0 (solvent) and 1 (protein)]. It makes limited use [e.g. `read(*,*)m`; `m=-m`; `p=minloc(m)`] of F-90 array functions:

```
integer::m(12,10,8),p(3),c(960)
read(*,*)m; m=-m; n=0
1 p=minloc(m); if (m(p(1),p(2),p(3)).ge.0) goto 7
n=n+1; c(n)=1; m(p(1),p(2),p(3))=n
2 l=0; do 5 k=1,8; do 4 j=1,10; do 3 i=1,12
if (m(i,j,k).ge.0) goto 3
if (max(m(mod(i,12)+1,j,k),m(mod(i+10,12)+1,j,k), &
m(i,mod(j,10)+1,k),m(i,mod(j+8,10)+1,k), &
m(i,j,mod(k,8)+1),m(i,j,mod(k+6,8)+1)).le.0) goto 3
m(i,j,k)=n; c(n)=c(n)+1; l=l+1
3 continue; 4 continue; 5 continue
if (l.gt.0) goto 2; goto 1
6 format(' Domain',i4,' contains',i5,' pixels')
7 write(*,6) (i,c(i),i=1,n); end
```

## Simple example – mean and variance

$$\bar{x} = \sum x / N \quad V = \sum (x - \bar{x})^2 / N$$

This appears to require two scans through the data: the first to find  $\bar{x}$  and the second to find  $V$ . A 'one-scan' method would be faster if the data are stored on a disk (or are in RAM but do not all fit into the CPU 'cache' at the same time). We will look into this trivial example in detail because it illustrates several important points.

## Fortran-90

Despite the massive campaigns for python etc., we should not forget that Fortran was specifically designed for scientific number crunching applications, and – especially Fortran-90 – still has many advantages for crystallographic calculations:

1. Fortran code is extremely portable and stable – the original SHELX-76 code still compiles and runs correctly on any modern computer without any changes being necessary.
2. It is easy to produce robust stand-alone statically-linked binaries with **ZERO DEPENDENCIES** (e.g. SHELX).
3. Many highly optimized and debugged libraries are available for numerical applications (e.g. the Intel MKL, which includes multithreaded multidimensional mixed-radix FFTs).
4. Fortran-90 added many useful new features, e.g. array and character operations, runtime memory allocation etc.
5. OPEN-MP enables multithreaded code to be generated easily.

## What is an algorithm ?

An algorithm is a way of calculating something that even the person who invented it cannot understand !?

Usually algorithms involve some clever mathematics to do something faster – sometimes by orders of magnitude – or 'better' than the obvious way of doing it.

Examples:

Sorting (e.g. reflection and peak lists)

FFT for structure factor calculations etc.

Fast calculation of interatomic distances

Derivation of phase relations for direct methods

## A one-scan algorithm

$$V = \sum (x - \bar{x})^2 / N = \sum (x^2 - 2x\bar{x} + \bar{x}^2) / N$$

But  $\bar{x} = \sum x / N$ . Substituting this:

$$V = \sum x^2 / N - [(\sum x) / N]^2$$

Which only requires one scan, summing both  $\sum x$  and  $\sum x^2$ .

Usually we need  $\sqrt{V}$ , so it is important to know whether it is ever possible for  $V$  calculated in the above way to be negative, this would 'crash' the program! In a mathematical sense  $V$  can never be negative, but it could still happen as a result of rounding errors if all  $x$  are equal and non-integral. So a good 'defensive' programmer would intuitively avoid this by replacing a negative value of  $\sum x^2 / N - [(\sum x) / N]^2$  by zero.

## The hidden trap

If the floating point numbers are (as is often the case) being stored in 4 bytes each, they will have a precision of about  $6\frac{1}{2}$  decimal digits. If we happen to be summing over  $10^8$  pixels of a large Fourier map to calculate the 'one sigma level'  $\sqrt{V}$ , both formulas will give the wrong answers !!

The reason is that when we add (say) the  $10000000^{\text{th}}$  pixel to the running totals, the increment will be smaller than the precision to which the numbers are being stored, so the running totals will not change, i.e. the increment is thrown away.

Possible remedies are to sum rows and layers of the map first and then add the totals, or do all the calculations with 8-byte (64-bit) numbers.

## Sort algorithms

Sorting algorithms are the highlight of many informatics courses because the difference in speed can be enormous. An obvious method such as scanning  $A(1..N)$  to find the largest element, then swapping it with  $A(1)$ , then scanning  $A(2..N)$  and swapping the largest element with  $A(2)$ , then scanning  $A(3..N)$  etc. takes a time of order  $N^2$  (which for large  $N$  can be all week). The best general algorithms are of order  $N \log N$  but are complicated (best to call a library routine).

Sometimes – as with sorting reflection lists – we can take advantage of the special features of the particular problem to reduce the order to  $N$ . For 10000 reflections,  $N^2$  is 100000000,  $N \log N$  is 50000 but  $N$  is only 10000 (but the constant factor multiplying the order may differ)!

## A typical sort algorithm

The following FORTRAN routine (called **comb-sort**) is a good general purpose algorithm for sorting a few hundred items (e.g. Fourier map peaks). Although not quite  $N \log N$  it is fast because it has a low overhead. The array  $A(1..N)$  (containing e.g. peak

```

K=N
1  K=INT(REAL(K)/1.2796)
  IF (K.LT.1) K=1
  IF (K.EQ.9.OR.K.EQ.10) K=11
  M=0
  DO 2 I=1,N-K
    J=I+K
    IF (A(J).GT.A(I)) THEN
      Q=A(J)
      A(J)=A(I)
      A(I)=Q
      M=1
    ENDIF
2  CONTINUE
  IF (M+K.GT.1) GOTO 1

```

heights) is sorted into descending order. In practice the IF..ENDIF loop would also need to swap the atom coordinates  $x$ ,  $y$  and  $z$  as well as the peak heights.

## Sorting reflection lists (order $N$ )

First  $h, k, l$  are transformed to a standard equivalent (e.g. maximum  $l$ , if  $l$  values are equal then maximum  $k$ , if both  $k$  and  $l$  equal then maximum  $h$ ). Then the maximum and minimum values of each index are found.

To sort on  $h$ , scan list, count how often each  $h$  is present, storing the results in an integer array  $N(h_{\min}..h_{\max})$ . This is then converted so that it holds 'pointers'  $p_h$  to the final list:

	$\downarrow p_3$	$\downarrow p_4$	$\downarrow p_5$	$\downarrow p_6$	FINAL LIST
...	all $h=3$	all $h=4$	all $h=5$	all $h=6$	...

The list is scanned again, putting each reflection into the final location pointed to by  $p_h$  and then incrementing  $p_h$ .

The list is sorted first on  $h$ , then on  $k$ , and finally on  $l$ . In the final sorted list, equivalents finish next to each other and so can easily be averaged.

## Sorting small integers (Fortran-77)

```

C
C SUBROUTINE INSORT (N,IP,IQ,ID)
C Sort-merge integer data in order of ascending ID(I). IP is the
C current pointer array to ID and IQ becomes the new pointer array.
C
      INTEGER :: IP(N), IQ(N), ID(N), IT(999)
      L=ID(1)
      M=L
      DO 1 I=2,N
        L=MINO(ID(I),L)
        M=MAXO(ID(I),M)
1      CONTINUE
      L=L-1
      M=M-L
      DO 2 I=1,M
        IT(I)=0
2      CONTINUE
      DO 3 I=1,N
        J=ID(I)-L
        IT(J)=IT(J)+1
3      CONTINUE
      J=0
      DO 4 I=1,M
        K=J
        J=J+IT(I)
        IT(I)=K
4      CONTINUE
      DO 5 I=1,N
        J=ID(IP(I))-L
        IT(J)=IT(J)+1
        IQ(IT(J))=IP(I)
5      CONTINUE
      RETURN
      END

```

## Sorting and merging reflections

Kevin Cowtan's notes on Symmetry in Reciprocal Space must be read before attempting this exercise!

The file "in" contains cell, symops and reflection data in free format for a small protein in space group  $P3_121$ . There are 389596 data before merging. The exercise is to sort-merge the data, remove systematic absences (and print them out with  $\| \sigma$  ratios) and count the number of unique reflections remaining, and how many of them are in centric projections (and so have no anomalous differences).  $R(\text{int}) = \sum |I - \langle I \rangle| / \sum I$  should also be calculated (only reflections in groups of more than one equivalent should be included in the  $R(\text{int})$  calculation).

There should be 42827 remaining unique reflections, 4354 of them centric and  $R(\text{int})$  is 0.0466.

Any computer language and libraries may be used.

## The symmetry operators

All symmetry-dependent information in real or reciprocal space can be derived from the symmetry operators! E.g. Space group P3<sub>1</sub>:

$$m=1: x, y, z; \quad m=2: -y, x-y, z+1/3; \quad m=3: -x+y, -x, z+2/3$$

These operators may also be expressed as 3x3 matrices R plus vectors t:

$$\begin{pmatrix} x_m \\ y_m \\ z_m \end{pmatrix} = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}$$

Or:  $x_m = R_m x + t_m$ , which in the case of operator m=3 is:

$$\begin{pmatrix} x_m \\ y_m \\ z_m \end{pmatrix} = \begin{pmatrix} -1 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 2/3 \end{pmatrix}$$

## Properties of R and t

The **determinant** of the matrix R must be +1 or -1. If it is -1 it produces an inverted image, so the space group is **not chiral** (but may still be non-centrosymmetric).

When one row of R is never negative for any operator [e.g. the third row (R<sub>31</sub> R<sub>32</sub> R<sub>33</sub>) in P3<sub>1</sub>] the space group is **polar**.

If all elements of t are zero for all operators (not including lattice centering) the space group is **symmorphic** (and has no systematic absences apart from lattice absences).

If t includes elements that are not multiples of 1/2 and the lattice is primitive the space group is **one member of an enantiomorphic pair**; if either all elements of t are multiples of 1/2 or the lattice is centered the space group does not belong to an enantiomorphic pair (!)

## Symmetry operators for P4<sub>1</sub>2<sub>1</sub>2

For the examples in this talk we will use the space groups P3<sub>1</sub> and P4<sub>1</sub>2<sub>1</sub>2; for the latter the general positions are:

$$\begin{array}{ll} m=1: x, y, z & m=2: -x, -y, z+1/2 \\ m=3: 1/2-y, 1/2+x, z+1/4 & m=4: -y, -x, 1/2-z \\ m=5: 1/2+y, 1/2-x, z+3/4 & m=6: 1/2-x, 1/2+y, 1/4-z \\ m=7: 1/2+x, 1/2-y, 3/4-z & m=8: y, x, -z \end{array}$$

i.e. for m=5:

$$R = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad t = \begin{pmatrix} 1/2 \\ 1/2 \\ 3/4 \end{pmatrix}$$

To obtain the general positions of the enantiomorphic space group P4<sub>3</sub>2<sub>1</sub>2, just exchange 1/4 and 3/4 !

## Symmetry in reciprocal space

For the symmetry operator m:  $x_m = R_m x + t_m$

The calculated structure factor F<sub>c</sub> is given by the complex number F<sub>c</sub> = (A + iB) where:

$$A_{hkl} = \sum_{\text{atoms}} \sum_{\text{symm}} f_j \cos[2\pi(hx_m + ky_m + lz_m)]$$

$$B_{hkl} = \sum_{\text{atoms}} \sum_{\text{symm}} f_j \sin[2\pi(hx_m + ky_m + lz_m)]$$

(the exponential term for atomic displacements has been included in the scattering factor f<sub>j</sub> here for simplicity). But

$$hx_m + ky_m + lz_m = h(R_m x + t_m) = h_m x + k_m y + l_m z + ht_1 + kt_2 + lt_3$$

where:  $h_m = R_{11}h + R_{21}k + R_{31}l$

$$k_m = R_{12}h + R_{22}k + R_{32}l$$

$$l_m = R_{13}h + R_{23}k + R_{33}l$$

So to find the **equivalent indices** h<sub>m</sub>, k<sub>m</sub>, l<sub>m</sub> we multiply h, k, l by the **transpose** of the matrix R.

## The phases of equivalent reflections

The phase  $\phi_m$  of the equivalent reflection h<sub>m</sub> is derived from the phase  $\phi$  of the (prime) reflection h by:

$$\phi_m = \phi - 2\pi h t_m = \phi - 2\pi (h t_1 + k t_2 + l t_3)$$

For example in P3<sub>1</sub>:

$$\begin{aligned} h_2 &= 0.h + 1.k + 0.l = k \\ k_2 &= -1.h - 1.k + 0.l = -h - k \\ l_2 &= 0.h + 0.k + 1.l = l \end{aligned}$$

So h<sub>2</sub> is k, -h-k, l with phase:

$$\phi_2 = \phi - 2\pi h t_2 = \phi - 2\pi (h.0 + k.0 + l.1/3) = \phi - 2\pi l/3$$

These are the true equivalent reflections; they have the same intensities and exactly the above phase shifts whether anomalous scatterers (that would cause Friedel's law to break down) are present or not.

## Friedel's law

Friedel's law states that  $|F_{-m}| = |F_m|$  and  $\phi_{-m} = -\phi_m$  where  $\phi_{-m}$  is the phase of  $-h_m - k_m - l_m$ . Friedel's law is strictly valid only when f'' is equal (or zero) for all atoms in the structure, but it is almost always a good approximation. In space group P3<sub>1</sub>:

$$|F_{h,k,l}| = |F_{k,-h-k,l}| = |F_{-h-k,h,l}| \quad (\text{exact equivalents}) \quad \text{and}$$

$$|F_{-h,-k,-l}| = |F_{-k,h+k,-l}| = |F_{h+k,-h,-l}| \quad (\text{exact equivalents})$$

but these two groups are only approximately equal because they are related by Friedel's law. For non-centrosymmetric space groups (chiral or not) there are always two groups of exact equivalents; if Friedel's law holds, the |F| values of the two groups are also the same.

## Equivalents in P4<sub>1</sub>2<sub>1</sub>2

For P4<sub>1</sub>2<sub>1</sub>2 the two groups of equivalents are:

$$\begin{aligned} h,k,l &= -h,-k,l = k,-h,l = -k,-h,l = -k,h,l = -h,k,l = h,-k,l = k,h,-l \\ -h,-k,-l &= h,k,-l = -k,h,-l = k,-h,-l = h,-k,-l = -h,k,-l = -k,-h,-l \end{aligned}$$

The space group P4mm has the same Laue group as P4<sub>1</sub>2<sub>1</sub>2 but different Friedel-related groups:

$$\begin{aligned} h,k,l &= -h,-k,l = k,-h,l = -k,-h,l = -k,h,l = -h,k,l = h,-k,l = k,h,l \\ -h,-k,-l &= h,k,-l = -k,h,-l = k,-h,-l = h,-k,-l = -h,k,-l = -k,-h,-l \end{aligned}$$

To derive these groups of equivalents correctly, it is necessary to know the point group (or space group). The Laue group contains an inversion center and so is not sufficient. It should be noted that for chiral compounds (i.e. for macromolecules) there is only one possible point group (the one that has rotation axes but no mirror planes, inversion centers or inverse tetrads) corresponding to each Laue group.

## Symmetry-restricted phases

If  $h_{-m} = h$  but  $\phi_{-m}$  is not equal to  $\phi (+2n\pi)$  then:

$$\phi_{-m} = -(\phi - 2\pi h t_m) = \phi (+2n\pi)$$

which gives the equation:

$$\begin{aligned} 2\phi &= 2\pi h t_m + 2n\pi \quad (n \text{ integer}) \\ \text{or } \phi &= \pi (h t_1 + k t_2 + l t_3 + n) \end{aligned}$$

So there can only be two possible values for  $\phi$  (corresponding to  $n$  odd and  $n$  even) and they must differ by  $\pi$ . Such reflections belong to a *centrosymmetric projection*. In P3<sub>1</sub>, for no values of  $m$  and  $h,k,l$  (except 0,0,0) is  $h_{-m} = h$ , so there are no centrosymmetric projections. This is clearly also true in real space from inspection of the IT diagram.

## Datafile for sort-merge exercise

The file "in" begins with a comment line, followed by the cell, symmetry and reflection data all in free format.

```
Trigonal bovine trypsin P3121 #152 CuKa
54.735 54.735 106.786 90 90 120
6 symops follow, then h,k,l,I and sig(I)
1 0 0 0 1 0 0 0 1 0.0 0.0 0.0
0 -1 0 1 -1 0 0 0 1 0.0 0.0 0.333333
-1 1 0 -1 0 0 0 0 1 0.0 0.0 0.666667
0 1 0 1 0 0 0 0 -1 0.0 0.0 0.0
1 -1 0 0 -1 0 0 0 -1 0.0 0.0 0.666667
-1 0 0 -1 1 0 0 0 -1 0.0 0.0 0.333333
22 -3 -41 21.38 3.27
-2 6 -12 162.92 11.71
-19 4 -32 81.44 6.82
-13 -9 -51 16.44 3.87
```

etc. 389596 reflections in total, terminated by the end of the file.

## Systematic absences

A reflection is *systematically absent* when  $h_m = h$  but  $\phi_m$  is not equal to  $\phi (+2n\pi)$  where  $n$  is an integer). In P3<sub>1</sub>:

$$\phi_{k,-h-k,l} = \phi_{h,k,l} - 2\pi l/3 (+2n\pi)$$

so when  $h = k = 0$ :

$$\phi_{0,0,l} = \phi_{0,0,l} - 2\pi l/3 (+2n\pi)$$

which can only be true when  $l = 3n$ , i.e. reflections 0,0, $l$  with  $l$  not equal to  $3n$  are *systematically absent*.

Note that the reflection is absent if this applies for *any* operator  $m$ . E.g. in P4<sub>1</sub>2<sub>1</sub>2:

$$m=2: \quad \phi_{-h,-k,l} = \phi_{h,k,l} - \pi l (+2n\pi)$$

which implies 0,0, $l$  absent for  $l$  not equal to  $2n$ , but:

$$m=3: \quad \phi_{k,-h,l} = \phi_{h,k,l} - \pi h - \pi k - \frac{1}{2}\pi l (+2n\pi)$$

which requires 0,0, $l$  absent for  $l$  not  $4n$ , so 0,0,2 is also absent.

## Centric reflections in P4<sub>1</sub>2<sub>1</sub>2

P4<sub>1</sub>2<sub>1</sub>2 has several classes of reflections with restricted phases, e.g.:

$$m=2: \quad h_{-m} = -(-h,-k,l)$$

which is equal to  $h,k,l$  when  $l=0$ , which gives:

$$\phi_{h,k,0} = \pi (h \cdot 0 + k \cdot 0 + 0 \cdot \frac{1}{2}) + n\pi = n\pi$$

so the  $h,k,0$  reflections have phases restricted to 0 or  $\pi$ .

Similarly,  $m=6$  gives  $h_{-m} = -(-h, k, -l)$  which is equal to  $h,k,l$  if  $k=0$ . Then:

$$\phi_{h,0,l} = \pi (h \cdot \frac{1}{2} + 0 \cdot \frac{1}{2} + l \cdot \frac{1}{4}) + n\pi = \pi (\frac{1}{2}h + \frac{1}{4}l) + n\pi$$

So for example the reflection 1,0,1 has two possible phases of  $3\pi/4$  or  $7\pi/4$ .

In the case  $m=4$ ,  $h_{-m} = -(-k,-h,-l)$  which is equal to  $h,k,l$  when  $h=k$ . Thus it can be shown that reflections  $h,h,l$  are restricted to  $\pi/2$  or  $3\pi/2$ .

## Hints for the tutorial

1. Remember to use *transposed* symops for transforming the reflections to their equivalents! Do not forget Friedel's law.
2. The basic idea is to transform all reflections to a standard setting (e.g. maximum  $l$ , if  $l$  is equal for two equivalents then maximum  $k$  etc.). This list of standardized reflections is then sorted so that reflections with the same indices appear adjacent in the sorted list. It is quicker to sort pointers than shuffling the contents of the lists.
3. Reflections with same indices are merged:  $\langle I \rangle = \sum I / n$ ;  $\sigma(\langle I \rangle) = 1 / [\sum (1/\sigma^2(I))]^{1/2}$  [for more sophisticated merging see Blessing, *J. Appl. Cryst.* 30 (1997) 421-426].
4. To find out if a reflection is systematically absent, generate equivalent  $h,k,l$  and see if any are identical to the original. A non-integral phase shift (calculated using  $h,k,l$  and the translational part of the symop.) then indicates an absence. Centric reflections have equivalents with indices  $-h,-k,-l$ .

## Fortran-77 and C++ solutions to exercise

Solutions to the exercise are provided in both Fortran-77 (*smerge.f*) and C++ (*sortmerge.cpp*, written by Tim Grüne). Note that the C++ file-reading routine could be replaced by the appreciably faster C-routine in the latter. Under Linux they may be compiled and run as follows:

<code>f77 smerge.f -o smerge</code>		<code>g++ sortmerge.cpp -o sortmerge</code>
<code>time ./smerge &lt;in</code>		<code>time ./sortmerge in</code>

which gave runtimes of 2.3 and 2.8 seconds resp. on a 3 GHz Xeon. After optimization (in both cases with `-O -ffast-math`) these were reduced to 1.9 and 2.3 seconds. The Intel ifort compiler was (in this somewhat untypical example) a little slower.

```

C
PROGRAM SMERG
C
C Fortran-77 sort-merge solution for Siena exercise
C
PARAMETER (NX=200000)
INTEGER IH (NX), IK (NX), IL (NX), IP (NX), IQ (NX)
REAL FF (NX), SI (NX), SY (12,24)
C
C Read data from standard input
C
READ (*, '( / ) ')
READ (*, *) NS
READ (*, *) ((SY (I, J), I=1, 12), J=1, NS)
NR=0
1 N=NR+1
IF (N.GT.NX) STOP '** Too many reflections **'
READ (*, *, END=5) IH (N), IK (N), IL (N), FF (N), SI (N)
IP (N)=N
NR=N
C
C Convert reflection indices to standard setting
C
U=REAL (IH (N))
V=REAL (IK (N))
W=REAL (IL (N))
DO 4 M=-1, 1, 2
DO 3 J=1, NS
I=M*NINT (SY (1, J) *U+SY (4, J) *V+SY (7, J) *W)
K=M*NINT (SY (2, J) *U+SY (5, J) *V+SY (8, J) *W)
L=M*NINT (SY (3, J) *U+SY (6, J) *V+SY (9, J) *W)
IF (L.LT.IL (N)) GOTO 3
IF (L.GT.IL (N)) GOTO 2
IF (K.LT.IK (N)) GOTO 3
IF (K.GT.IK (N)) GOTO 2
IF (I.LE.IH (N)) GOTO 3
2 IH (N)=I
IK (N)=K
IL (N)=L
3 CONTINUE
4 CONTINUE
GOTO 1
C
C Sort pointer arrays on h, then k, then l
C
5 CALL INSORT (NR, IP, IQ, IH)
CALL INSORT (NR, IQ, IP, IK)
CALL INSORT (NR, IP, IQ, IL)
C
C Combine equivalents - now contiguous
C
R=0.
S=0.
NU=NR
NC=0
N=1
6 IF (N.GT.NR) GOTO 16
M=N
MQ=IQ (M)
P=0.
Q=0.
7 NQ=IQ (N)
IF (IH (NQ) .NE. IH (MQ) .OR. IK (NQ) .NE. IK (MQ) .OR. IL (NQ) .NE. IL (MQ)) GOTO 8
P=P+FF (NQ)
Q=Q+1./SI (NQ) **2
N=N+1
IF (N.LE.NR) GOTO 7

```

```

8   P=P/REAL (N-M)
   Q=1./SQRT (Q)
C
C Detect systematic absences and centric reflections
C
   U=REAL (IH (MQ) )
   V=REAL (IK (MQ) )
   W=REAL (IL (MQ) )
   NT=0
   DO 13 J=2,NS
     I=NINT (SY (1,J) *U+SY (4,J) *V+SY (7,J) *W)
     K=NINT (SY (2,J) *U+SY (5,J) *V+SY (8,J) *W)
     L=NINT (SY (3,J) *U+SY (6,J) *V+SY (9,J) *W)
     IF (I.NE.IH (MQ) .OR.K.NE.IK (MQ) .OR.L.NE.IL (MQ) ) GOTO 12
     IF (MOD (NINT (12.* (SY (10,J) *U+SY (11,J) *V+SY (12,J) *W) ),12) .EQ.0)
11  + GOTO 13
     FORMAT (' Reflection',3i4,' syst. abs., I/sigma =',F8.2)
     WRITE (*,11) IH (MQ) , IK (MQ) , IL (MQ) , P/Q
     GOTO 6
12  IF (-I.EQ.IH (MQ) .AND.-K.EQ.IK (MQ) .AND.-L.EQ.IL (MQ) ) NT=1
13  CONTINUE
   NC=NC+NT
   NU=NU+1
   IF (NU.GT.NX) STOP '** Too many reflections **'
   IH (NU) = IH (MQ)
   IK (NU) = IK (MQ)
   IL (NU) = IL (MQ)
   FF (NU) = P
   SI (NU) = Q
   IF (N.LE.M+1) GOTO 6
   DO 14 I=M,N-1
     R=R+ABS (FF (IQ (I) ) -P)
     S=S+P
14  CONTINUE
   GOTO 6
C
C Unique reflections now in IH(NR+1..NU) etc. Output statistics.
C
15  FORMAT (/I6,' Unique reflections, of which',I5,' centric ',
+ 'R(int) =',F8.4/)
16  WRITE (*,15) NU-NR,NC,R/S
   END
C
C -----
C
   SUBROUTINE INSORT (N,IP,IQ,ID)
C
C Sort-merge integer data in order of ascending ID(I). IP is the
C current pointer array to ID and IQ becomes the new pointer array.
C
   INTEGER IP (N) , IQ (N) , ID (N) , IT (9999)
   L=ID (1)
   M=L
   DO 1 I=2,N
     L=MIN0 (ID (I) , L)
     M=MAX0 (ID (I) , M)
1  CONTINUE
   L=L-1
   M=M-L
   DO 2 I=1,M
     IT (I) =0
2  CONTINUE
   DO 3 I=1,N
     J=ID (I) -L
     IT (J) =IT (J) +1
3  CONTINUE
   J=0

```



```
      DO 4 I=1,M
      K=J
      J=J+IT(I)
      IT(I)=K
4      CONTINUE
      DO 5 I=1,N
      J=ID(IP(I))-L
      IT(J)=IT(J)+1
      J=IT(J)
      IQ(J)=IP(I)
5      CONTINUE
      RETURN
      END
```

```

/**
 * \file      sort_reflections.cpp
 * \date      16/08/2005
 * \author    Tim Gruene
 * \read in a file with header (3 + n(symops) lines) and sort reflections,
 * \first by h, then by k, then by l
 */

#include <algorithm>
#include <iostream>
#include <iomanip>
#include <cmath>
#include <vector>
#include <fstream>
#include <sstream>
#include <string>

// definition of a useful struct
struct Symop
{
    int R[3][3];
    int T[3];
};

class Reflection
{
private:
    int h_, k_, l_;
    float I_, sigI_;
    bool absent;
public:
    Reflection(int h, int k, int l, float I, float sigI):
        h_(h), k_(k), l_(l), I_(I), sigI_(sigI), absent(false){}
    ~Reflection(){}

    // retrieve data members
    int h() const { return h_; }
    int k() const { return k_; }
    int l() const { return l_; }
    float I() const { return I_; }
    float sigI() const { return sigI_; }

    // equality operator -- based on indices only
    bool operator==(const Reflection& r) const
    { return (h_ == r.h_) && (k_ == r.k_) && (l_ == r.l_); }

    // comparison operator
    inline bool operator<(const Reflection&) const;

    // multiplication with Symop
    inline Reflection operator*(const Symop&) const;

    //inline Reflection operator*(int) const;
    //! negation operator
    inline Reflection operator-() const;

    // set absence flag
    void set_absence() { absent=true; }

    // returns sym.-equivalent with maximal index
    Reflection max_equivalent(std::vector<Symop>&) const;

    // check absence
    friend bool is_absent(const Reflection& r) { return r.absent; }

    // for printing
    friend std::ostream& operator<<(std::ostream& os, const Reflection&);

```

```

};

typedef std::vector<Reflection> Reflexes;

// forward declarations of functions called from main
void usage();
int read_reflections(char *, std::vector<Symop>&, std::vector<Reflection>&);
int merge_reflections(const Reflexes&, Reflexes&);
int remove_sys_abs(const std::vector<Symop>&, Reflexes&);
int sys_absences(const std::vector<Symop>&, Reflexes&);
float Rint(const std::vector<Symop>&, const Reflexes&, const Reflexes&);

int main(int argc, char* argv[])
{
    std::vector<Reflection> reflexes, refl_merged;
    std::vector<Symop> symops;
    std::vector<Reflection>::iterator it;

    if ( argc < 2 )
    {
        usage();
        return -1;
    }

    if (read_reflections(argv[1], symops, reflexes))
    {
        std::cerr << "An error occurred while reading " << argv[1] << '\n';
        return EXIT_FAILURE;
    }

    std::cout << "Standardising indices.\n";
    for ( it = reflexes.begin(); it != reflexes.end(); it++)
    {
        *it = it->max_equivalent(symops);
    }

    std::cout << "Sorting reflections.\n";
    std::sort(reflexes.begin(), reflexes.end());

    std::cout << "Merging symmetry equivalents.\n";
    merge_reflections(reflexes, refl_merged);

    std::cout << "Analysing systematic absences.\n";
    sys_absences(symops, refl_merged);

    std::cout << "Number of reflections: " << reflexes.size() << '\n';

    std::cout << "After merging, there are " << refl_merged.size()
        << " unique reflections.\n";

    std::cout << "R(int) = "
        << std::fixed << std::setprecision(4)
        << Rint(symops, reflexes, refl_merged) << '\n';

    return 0;
}

// compare hkl indices of two reflections
bool Reflection::operator<(const Reflection& other) const
{
    if ( h_ < other.h_ ) return true;
    else if ( h_ > other.h_ ) return false;
    else // h1 == h2
    {
        if ( k_ < other.k_ ) return true;
        else if ( k_ > other.k_ ) return false;
    }
}

```

```

        else // k1 == k2
        {
            if ( l_ < other.l_ ) return true;
            else if ( l_ > other.l_ ) return false;
        }
    }

    // h1 == h2 && k1 == k2 && l1 == l2
    return false;
}

// finds the equivalent with maximal indices
Reflection Reflection::max_equivalent(std::vector<Symop>& symops) const
{
    Reflection max_refl = *this;
    std::vector<Symop>::const_iterator it;

    for ( it = symops.begin(); it != symops.end(); it++)
    {
        Reflection sym_refl = ((*this)* (*it));

        if ( max_refl < sym_refl )
        {
            max_refl = sym_refl;
        }
        else if ( max_refl < -sym_refl )
        {
            max_refl = -sym_refl;
        }
    }
    return max_refl;
}

Reflection Reflection::operator*(const Symop& symop) const
{
    Reflection produkt(*this);

    produkt.h_ = h_*symop.R[0][0] + k_*symop.R[1][0] + l_*symop.R[2][0];
    produkt.k_ = h_*symop.R[0][1] + k_*symop.R[1][1] + l_*symop.R[2][1];
    produkt.l_ = h_*symop.R[0][2] + k_*symop.R[1][2] + l_*symop.R[2][2];

    return produkt;
}

/*
Reflection Reflection::operator*(int m) const
{
    Reflection r(*this);
    r.h_ = m*this->h_;
    r.k_ = m*this->k_;
    r.l_ = m*this->l_;

    return r;
}
*/

/**
 * returns a reflex with negated indices
 */
Reflection Reflection::operator-() const
{
    Reflection neg(*this);
    neg.h_ = -neg.h_;
    neg.k_ = -neg.k_;
    neg.l_ = -neg.l_;

    return neg;
}

```

```

}

std::ostream& operator<<(std::ostream& os, const Reflection& r)
{
    os << std::fixed
        << std::setw(6) << r.h_
        << std::setw(6) << r.k_
        << std::setw(6) << r.l_
        << std::setw(10) << std::setprecision(3) << r.I_
        << std::setw(10) << std::setprecision(3) << r.sigI_;
    return os;
}

void usage()
{
    std::cout << "Usage: siena <filename>.\n";
}

/**
 * read a list of reflections from filename. Format must be:
 * line 1: title/comment
 * line 2: cell ( a b c alpha beta gamma)
 * line 3: n symops ( n = number of symops)
 * line 4 - 4+n: symops
 */
int read_reflections(char * filename,
                    std::vector<Symop>& symops, std::vector<Reflection>& refls)
{
    std::ifstream input(filename);
    std::string line; // for getline
    int num_symops;

    if (! input.is_open())
    {
        std::cout << "Could not open file " << filename << '\n';
        return -1;
    }

    // skip first two lines
    std::getline(input, line);
    std::getline(input, line);

    // get number of symops
    std::getline(input, line);
    std::istringstream num_symops_stream(line);

    num_symops_stream >> num_symops;

    // read in symops
    for ( int i = 0; i < num_symops; i++)
    {
        std::getline(input, line);
        std::istringstream symop_stream(line);

        Symop symop;
        float tx, ty, tz;

        for ( int j = 0; j < 3; j++)
            for ( int k = 0; k < 3; k++)
                symop_stream >> symop.R[j][k];
        symop_stream >> tx >> ty >> tz;

        symop.T[0] = int ( 0.5+12*tx);
        symop.T[1] = int ( 0.5+12*ty);
        symop.T[2] = int ( 0.5+12*tz);
    }
}

```

```

        symops.push_back(symop);
    }

    // now read in reflections
    refls.clear();

    while ( true )
    {
        int h, k, l;
        float I, sigI;
        input >> h >> k >> l >> I >> sigI;
        if(input.eof()) break;
        refls.push_back(Reflection(h,k,l,I,sigI));

        /*
        std::istringstream refl_stream(line);

        refl_stream >> h >> k >> l >> I >> sigI;
        refls.push_back(Reflection(h,k,l,I,sigI));
        // get next line
        std::getline(input, line);
        */
    }

    return 0;
}

/**
 * Merges list of Reflections unmerged based on the list of symmetry operators
 * and puts the merged list into merged
 */
int merge_reflections(const Reflexes& unmerged, Reflexes& merged)
{
    Reflexes::const_iterator it;

    for ( it = unmerged.begin(); it != unmerged.end(); it++)
    {
        double sumI      = 0.0;
        double sum_sigma = 0.0;
        int    num_sym_equiv = 0;

        Reflection reference(*it);

        while ( reference == *it )
        {
            sumI      += it->I();
            sum_sigma += 1.0/(it->sigI() * it->sigI());
            ++num_sym_equiv;
            ++it;
        }
        merged.push_back(Reflection(reference.h(), reference.k(), reference.l(),
                                     sumI/num_sym_equiv, 1.0/std::sqrt(sum_sigma)));
        // rewind by one
        --it;
    }

    return 0;
}

/**
 * checks merged for systematically absent reflections and prints I/sigI for
 * these reflections. They are removed from the list. Also counts number of
 * centric reflections
 * \param  symops  list of symmetry operators
 * \param  merged  list of merged reflexes
 */

```

```

int sys_absences(const std::vector<Symop>& symops, Reflexes& merged)
{
    Reflexes::iterator it_r;
    std::vector<Symop>::const_iterator it_sym;
    int num_sys_abs(0);
    int centric_reflections(0);
    bool is_centric;

    for ( it_r = merged.begin(); it_r != merged.end(); it_r++)
    {
        is_centric = false;
        for ( it_sym = symops.begin()+1; it_sym != symops.end(); it_sym++)
        {
            Reflection equiv = (*it_r)*(*it_sym);
            if ( equiv == (*it_r) )
            {
                if ( (!it_sym->T[0]) && (!it_sym->T[1]) && (!it_sym->T[2]) )
                {
                    continue;
                }
                else
                {
                    int shift = (equiv.h() * it_sym->T[0] +
                                equiv.k() * it_sym->T[1] +
                                equiv.l() * it_sym->T[2]);
                    if ( shift%12 ) // the remainder of division
                    {
                        it_r->set_absence();
                        std::cout << "Reflection " << *it_r
                                  << " syst. abs., I/sigI: "
                                  << std::fixed
                                  << std::setw(7)
                                  << std::setprecision(2)
                                  << it_r->I()/it_r->sigI()
                                  << '\n';
                        break;
                    }
                }
            }
            else // indices are not identical
            {
                if ( equiv == -(*it_r) ) // check for centric reflex
                {
                    is_centric = true;
                }
                continue;
            }
        }
        if ( is_centric ) ++centric_reflections;
    }

    // Now let's remove absent reflections from the list
    Reflexes::iterator it_remove;
    it_remove = std::remove_if(merged.begin(), merged.end(), is_absent);
    merged.erase(it_remove, merged.end());

    std::cout << "Number of centric reflections: " << centric_reflections
              << std::endl;
    return (num_sys_abs);
}

/**
 * Calculate R_int for list unmerged, <I> is taken from merged
 * \param symops list of symmetry operators
 * \param unmerged list of sorted but unmerged reflections
 * \param merged list of merged reflections ( for <I>
 */

```



```

float Rint(const std::vector<Symop>& symops, const Reflexes& unmerged,
          const Reflexes& merged)
{
    Reflexes::const_iterator it;
    Reflexes::const_iterator it_Imean;
    double R_int (0.0);
    double I_total(0.0);

    it_Imean = merged.begin();

    for ( it = unmerged.begin(); it != unmerged.end(); it++)
    {
        int    num_sym_equiv(0);
        float  Imean;
        double r_int (0.0);
        double i_total(0.0);

        Reflection test(*it);

        // we could also simply increase it_Imean after each while-loop
        //it_Imean = std::find(it_Imean, merged.end(), test);
        Imean = it_Imean->I();
        if ( ! (*it_Imean == *it)) continue;
        ++it_Imean;

        while (test == *it && it != unmerged.end() )
        {
            r_int  += std::abs(it->I() - Imean);
            i_total += it->I();
            ++num_sym_equiv;
            ++it;
        }
        // only consider reflections with more than one symmetry mate
        if (num_sym_equiv > 1)
        {
            R_int  += r_int;
            I_total += i_total;
        }
        // while loop went one too far
        --it;
    }

    return (R_int / I_total);
}

```

```

#include "small_matrix.hpp"
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <vector>
#include <list>
#include <algorithm>
#include <valarray>

typedef Math::tMatrix<3> Matrix;
typedef Math::tVector<3> Vector;

class Reflection {
public:
    Vector hkl;
    float intensity;
    float sigma;
    float inverse_sqr_sigma;    // accumulates 1/sqare(sigma)
    float size;                // and n, for equivalent reflections

    bool centric;
    bool absent;
    float phase_shift;

    Reflection () {
        size=1;                // n=1 for each unmerged reflection
        centric = false;
        absent  = false;
    }

    // create a reflection from hkl, I and sigma etc
    Reflection (Vector v, float i, float s, bool ab=false, bool cen=false,
float shift = 0.0) {
        hkl      = v;
        intensity = i;
        sigma     = s;
        size      = 1.0;
        inverse_sqr_sigma = 1.0 / (s * s);
        absent    = ab;
        centric    = cen;
        phase_shift = shift;
    }

    // adds another reflection to this one if equivalent and return true
    // if not equivalent, just return false
    bool add(const Reflection & another_refl) {
        if ( this->hkl == another_refl.hkl ) {
            this->intensity += another_refl.intensity;
            this->inverse_sqr_sigma += another_refl.inverse_sqr_sigma;
            this->size += 1.0;

            this->centric = (this->centric || another_refl.centric);
            this->absent  = (this->absent || another_refl.absent);

            return true;
        }
        return false;
    }

    float merged_sigma() {
        return 1.0 / ( sqrt (inverse_sqr_sigma));
    }

    float merged_intensity() {
        return intensity / size;
    }
}

```

```

    float intensity_over_sigma() {
        return intensity / (size * sqrt (inverse_sqr_sigma));
    }

}; // Reflection

// comparison of reflections
bool operator < (const Reflection& lhs, const Reflection& rhs) {
    return (lhs.hkl < rhs.hkl);
}

// equality of reflections
bool operator == (const Reflection& lhs, const Reflection& rhs) {
    return (lhs.hkl == rhs.hkl);
}

// pretty printing of reflections
std::ostream& operator<<(std::ostream& os, Reflection& ref) {
    using namespace std;
    os<<setiosflags(ios::fixed);
    os<<setprecision(3);
    os<<"Reflection "<<ref.hkl<<" I/sigma =
"<<setw(8)<<ref.intensity_over_sigma();
    return os;
}

// A function object initialized with a lists of symops (rotational,
translational)
// Applied to each reflection,
// it translates each input hkl to the reduced hkl and returns it.
struct HKLReducer {
    std::vector< Matrix > Rt; // transpose of rotational
    std::vector< std::valarray<float> > Tr; // translational
    Vector hkl_m;
    int max;

    HKLReducer( const std::vector< Matrix > &rot ,
               const std::vector< std::valarray<float> >
&trans) {
        Rt = rot;
        Tr = trans;
        max = (rot.size() * 2)-1;
    }

    // this is where the magic happens
    Reflection operator() (const Reflection& ref) {
        const Vector& prime_hkl = ref.hkl;
        bool centric=false;
        bool absent = false;
        float phase_shift;

        std::vector< Vector > list_of_equivalents;
        list_of_equivalents.push_back(prime_hkl); // store prime HKL
and its friedel mate
        list_of_equivalents.push_back(-prime_hkl);

        // loop over rotational symops Rt_m and generate equivalent
hkl_m
        for (int m=1; m< Rt.size(); ++m) {
            hkl_m = Rt[m] * prime_hkl;
            list_of_equivalents.push_back(hkl_m); // add hkl_m and its
friedel mate in
            list_of_equivalents.push_back(-hkl_m); // to the list of
equivalents

```

```

        // identify systematic absences
        if (hkl_m == prime_hkl) {
            phase_shift = (
                (Tr[m])[0] * prime_hkl(0)
                + (Tr[m])[1] * prime_hkl(1)
                + (Tr[m])[2] * prime_hkl(2)
            );
            if (std::abs(phase_shift - round(phase_shift)) > 0.001)
                absent = true;
        } else if (hkl_m == -prime_hkl) {
            centric = true;
        }
    }

    // The maximum hkl in the list represents the bunch
    sort(list_of_equivalents.begin(), list_of_equivalents.end());

    return Reflection(list_of_equivalents[max],
                      ref.intensity, ref.sigma, absent, centric,
                      phase_shift);
}

};

class printHKL{
public:
    int absent;
    int centric;
    int rest;

    printHKL() {
        rest = centric = absent = 0;
    }

    void operator() (Reflection &r){
        if (r.absent) {
            ++absent;
            std::cout<<r<<std::endl;
        } else if (r.centric) {
            ++centric;
            ++rest;
        } else ++rest;
    }

    void complete() {
        std::cout<<std::endl<<rest<<" unique reflections of which "
                <<centric<<" are centric"<<std::endl;
        std::cout<<absent<<" systematic absences"<<std::endl;
    }

    ~printHKL() { complete(); }
} my_printer;

int main () {
    std::ifstream data_file ("in");
    std::string junk_string;
    int number_of_symops=6;    // symops to read from file

    std::vector< Matrix >          list_of_Rt;
    std::vector< std::valarray<float> > list_of_Tr;
    Matrix                        Rt(false);
    std::valarray<float> Tr(3);

    // Skip header
    for (int i=0 ; i < 3 ; ++i) getline (data_file,junk_string);

```

```

    for (int i=0 ; i < number_of_symops ; ++i) {
        // Read symops and store the transposes;
        for (int k=0; k < 9; ++k) {
            data_file >> Rt(k % 3, k / 3); // swap i and j to read in transpose
directly
        }

        data_file >>Tr[0]>>Tr[1]>>Tr[2];
        list_of_Rt.push_back( Rt );
        list_of_Tr.push_back( Tr );
    }

    // let the reducer know which symops to use.
    HKLReducer my_hkl_reducer(list_of_Rt, list_of_Tr);

    std::vector<Reflection> data_list;
    Vector tmp_hkl;
    float tmp_i, tmp_s;

    while (!data_file.eof()) {

        // read in line of data: h, k, l, int, sigma
        data_file>>tmp_hkl(0)>>tmp_hkl(1)>>tmp_hkl(2)>>tmp_i>>tmp_s;

        // transform Reflection to asym unit and store in data_list
        data_list.push_back( my_hkl_reducer(Reflection(tmp_hkl, tmp_i, tmp_s)
));
    }

    sort(data_list.begin(), data_list.end());

    // loop through data_list, add up multiple entries and transfer into new
    // vector

    std::vector<Reflection> merged_data;
    merged_data.push_back( data_list[0] ); // store first element before loop
    int my_pos = 0;
    for( int i=1; i < data_list.size(); ++i) {
        if ( ! merged_data[my_pos].add( data_list[i] ) ) {
            ++my_pos;
            merged_data.push_back( data_list[i]);
        }
    }

    for_each(merged_data.begin(), merged_data.end(), my_printer);
    my_printer.complete();
}

/*
// overload mult operator for multiplying matrix and vector
const Vector operator * (const Matrix& lhs,const Vector& rhs) {
    Vector tmp;
    for ( int i = 0; i < 3; ++i) {
        tmp(i) = 0;
        for ( int j = 0; j < 3; ++j)
            tmp(i) += static_cast<int>( lhs(i,j) * rhs(j) );
    }
    return tmp;
}

// a comparison operator is needed to sort a container of Vectors
bool operator < (const Vector& lhs, const Vector& rhs) {
    if (lhs(0) < rhs(0)) {
        return true;
    } else if (lhs(0) == rhs(0)) {
        if (lhs(1) < rhs(1)) {
            return true;

```

```
        } else if (lhs(1) == rhs(1)) {
            if (lhs(2) < rhs(2)) {
                return true;
            } else return false;
        } else return false;
    } else return false;
}

// test element-wise equality of vectors
bool operator == (const Vector& lhs, const Vector& rhs) {
    if (lhs(0) == rhs(0))
        if (lhs(1) == rhs(1))
            if (lhs(2) == rhs(2))
                return true;
    return false;
}
*/
```

```

#ifndef tMatrix_H
#define tMatrix_H

#include <ostream>
#include <iomanip>
#include <limits>

namespace Math {

template <class T>
inline T sign(T a, T b) {
    return (b) >= 0.0 ? fabs(a) : -fabs(a);
}

template <class T>
inline T sqr(T a) {
    return (a * a);
}

template <class T>
inline void swap(T& a, T& b) {
    T y = a;
    a = b;
    b = y;
}

template <int order>
class tMatrix {
protected:
    float mVal[order][order];
    int pos; // for operator , use only
public:
    static const tMatrix<order> identity() {
        tMatrix<order> a;
        for ( int i = 0; i < order; ++i)
            for ( int j = 0; j < order; ++j)
                a(i,j) = (i==j) ? 1 : 0;

        return a;
    }

    tMatrix<order>(bool initialize=true) {
        if (initialize) reset();
    }
    float& operator()( int i, int j) { return mVal[i][j]; }
    const float& operator()( int i, int j) const { return
mVal[i][j]; }

    tMatrix<order>& operator=(float d) {
        int i = pos / order;
        int j = pos % order;
        mVal[i][j] = d;
        pos++;
        return *this;
    }

    const tMatrix<order> transpose(){
        tMatrix<order> b;
        for ( int i = 0; i < order; ++i)
            for ( int j = i; j < order; ++j) {
                b(j,i)=mVal[i][j];
                b(i,j)=mVal[j][i];
            }
        return b;
    }

    void reset() {
        for ( int i = 0; i < order; ++i)

```



```

        for ( int j = 0; j < order; ++j)
            (*this)(i,j) = 0.0;
        pos = 0;
    }

};

template <int order>
class tVector {
public:
    int mVal[order];
    int pos; // for operator , use only
    float shift;
    bool centric;
    bool absent;

    tVector<order>(bool initialize=true) {
        if (initialize) reset();
        absent = false;
        centric = false;
    }

    int& operator()( int i) { return mVal[i]; }
    const int& operator()( int i) const { return mVal[i]; }
    void reset() {
        for ( int i = 0; i < order; ++i)
            (*this)(i) = 0;
        pos = 0;
    }

    const tVector<order> operator-() const {
        tVector<order> tmp;
        for ( int i = 0; i < order; ++i)
            tmp(i) = -mVal[i];
        return tmp;
    }

};

template <int order>
std::ostream& operator<<(std::ostream& os, const tMatrix<order>& m) {
    using namespace std;
    os<<setiosflags(ios::fixed);
    os<<setprecision(3);
    os<<order<<" x " <<order<<"=\n";
    for(int i=0;i<order;++i) {
        os<<"[";
        for(int j=0;j<order;++j)
            os<<setw(8)<<m(i,j)<<",";
        os<<"]\n";
    }
    return os;
}

template <int order>
std::ostream& operator<<(std::ostream& os, const tVector<order>& m) {
    using namespace std;
    os<<setiosflags(ios::fixed);
    os<<setprecision(3);
    for(int j=0;j<order;++j){
        os<<setw(5)<<m(j);
    }
    return os;
}

template <int order>

```

```

const tVector<order> operator * (const tMatrix<order>& lhs, const
tVector<order>& rhs) {
    tVector<order> tmp;
    for ( int i = 0; i < order; ++i) {
        tmp(i) = 0;
        for ( int j = 0; j < order; ++j)
            tmp(i) += static_cast<int>( lhs(i,j) * rhs(j) );
    }
    return tmp;
}

bool operator < (const tVector<3>& lhs, const tVector<3>& rhs) {
    if (lhs(0) < rhs(0)) {
        return true;
    } else if (lhs(0) == rhs(0)) {
        if (lhs(1) < rhs(1)) {
            return true;
        } else if (lhs(1) == rhs(1)) {
            if (lhs(2) < rhs(2)) {
                return true;
            } else return false;
        } else return false;
    } else return false;
}

bool operator == (const tVector<3>& lhs, const tVector<3>& rhs) {
    if (lhs(0) == rhs(0))
        if (lhs(1) == rhs(1))
            if (lhs(2) == rhs(2))
                return true;
    return false;
}

} // Math

#endif

```

```

Program DataRed
  use Crystallographic_Symmetry, only: Space_Group_Type, Set_SpaceGroup,
Write_SpaceGroup
  use String_Uutilities,          only: u_case
  use Reflections_Uutilities,     only: hkl_absent, hkl_equiv, hkl_s
  use Math_gen,                  only: sort, asind
  use Crystal_types,             only: Crystal_Cell_Type, Set_Crystal_Cell,
Write_Crystal_Cell

  implicit none

  integer, parameter              :: nref=400000, inp=1, ihkl=3, irej=4, iin=10,
i_scr=99
  integer, dimension(3,nref)     :: h
  integer, dimension(3)          :: h1,h2
  real,    dimension(nref)       :: intens, sigma, twtheta, intav, sigmav
  integer, dimension(nref)       :: itreat, iord, nequv, ini, fin, warn
  real,    dimension(48)        :: weight
  character(len=256)             :: filein, fileout, filecon
  character(len=132)             :: line, cmdline, title
  character(len=20)              :: spg_symb
  character(len=6)               :: key
  real, dimension(3)             :: cel,ang
  type (Space_Group_Type)        :: grp_especial
  type (Crystal_Cell_Type)       :: celda
  character(len=*) , parameter, dimension(0:1) :: warn_mess=("/"
", &
                                                                    " <- Dubious
reflection"/)
  Logical :: Friedel=.true., cell_given=.false., wave_given=.false.
  real    :: sig, suma, suman, Rint, &
          wavel,sigg, delt, warning, t_start, t_end
  integer :: i,j,k, ier, nr=0, ns, rej, len_cmdline, &
          lenf, nin, cent
  integer :: iargc, narg

!----- Treating the command line
  narg=iargc()
  len_cmdline=0
  if(narg > 0) then
    call getarg(1,cmdline)
    len_cmdline=len_trim(cmdline)
  end if

  if(len_cmdline /= 0) then
    lenf=index(cmdline," ") -1
    filecon=cmdline(1:lenf)//".red"
    open(unit=iin,file=filecon,status="old",iostat=ier,action="read")
    if(ier/=0) then
      write(unit=*,fmt="(3a)") " => File: ", trim(filecon)," not found!"
      stop
    end if
    read(unit=iin,fmt="(a)") title
  else
    write(unit=*,fmt="(a)") " => Please invoke the program as: 'data_red
myfile' where myfile.res is the input file"
    stop
  end if

!----- End Treating the command line

  call cpu_time(t_start)
  write(unit=*,fmt="(a)") " ====="
  write(unit=*,fmt="(a)") " DATA REDUCTION PROGRAM: DataRed"
  write(unit=*,fmt="(a)") " ====="
  write(unit=*,fmt="(a)") " "
  twtheta(:) =0.0

```

```

!----- Start reading the input command file

read(unit=iin,fmt="(a,a)") key, filein
filein=adjustl(filein)
write(unit=*,fmt="(a,a)") " => Name of the input file: ", trim(filein)
read(unit=iin,fmt="(a,a)") key, fileout
fileout=adjustl(fileout)
write(unit=*,fmt="(a,a)") " => Code of the output file: ", trim(fileout)
warning=0.30 ! 30% error for warning equivalent reflections

do
  read(unit=iin,fmt="(a)", iostat=ier) line
  if(ier /= 0) exit
  line=adjustl(line)
  if(line(1:1) == "!") cycle

  key=u_case(line(1:5))

  Select Case (key(1:5))

    Case ("SPGR ")
      spg_symb=adjustl(line(6:))

    Case ("NFRDL")
      Friedel=.false.

    Case ("CELL ")
      read(unit=line(7:),fmt=*) cel, ang
      call Set_Crystal_Cell(cel,ang,Celda)
      cell_given=.true.

    Case ("WAVE ")
      read(unit=line(7:),fmt=*) wavel
      wave_given=.true.

  End Select

end do

! check that all is O.K.
if(.not. cell_given) then
  write(unit=*,fmt=*) " => UNIT CELL not GIVEN! Modify your input file."
  stop
end if
if(.not. wave_given) then
  write(unit=*,fmt=*) " => WAVELENGTH not GIVEN! Modify your input file."
  stop
end if

!----- End reading the input command file

!----- Start reading the INTENSITY input file
open(unit=inp, file=filein, status="old", iostat=ier,action="read")
nr=0

!Reading reflections and calculate 2theta

do
  nr=nr+1
  read(unit=inp,fmt=*,iostat=ier) h1(:), intens(nr), sigma(nr)
  if(ier /= 0) then
    nr=nr-1
    exit
  end if
end do

```

```

    twtheta(nr)=2.0* ASIND( hkl_s(h1,celda)*WAVEL)
    if(twtheta(nr) < 0.0001) ier=1
    h(:,nr)=h1(:)
    if(sigma(nr) <= 0.0 ) sigma(nr)=0.004
end do

    write(unit=*,fmt="(a,i6)") " => Total number of reflections read: ", nr

!----- End reading the INTENSITY input file

!
! Order the reflections by ascending twtheta
!
    call sort(twtheta,nr,iord)

! Non-elegant way of ordering things
    open(unit=i_scr,status="scratch",form="unformatted",action="readwrite")
    do i=1,nr
        k=iord(i)
        write(unit=i_scr) h(:,k), intens(k), sigma(k), twtheta(k)
    end do
    rewind (unit=i_scr)
    do k=1,nr
        read(unit=i_scr) h(:,k), intens(k), sigma(k), twtheta(k)
    end do
    close(unit=i_scr)
    write(unit=*,fmt="(a)") " => Reflections ordered by ascending two-theta
O.K.!"
!
! Set symmetry
!
    call Set_SpaceGroup(spg_symb,grp_especial)

!
! Opening file for rejected reflections
!
    open(unit=irej, file=trim(fileout)//".rej", status="replace",action="write")
    write(unit=irej,fmt="(a)") " REJECTED REFLECTIONS (Symmetry
forbidden)"
    write(unit=irej,fmt="(a)") " h k l Intensity Sigma TwoTheta
I/sig"
    write(unit=irej,fmt="(a)") "
=====
!
! First loop over reflections
!
    nin=0
    itreat(:)=0
    ini(:)=0
    fin(:)=0

    rej=0

    do i=1,nr !Loop over all measured reflections

        if(itreat(i) == 0) then !If not yet treated do the following
            h1(:)=h(:,i)
            if(hkl_absent(h1,grp_especial)) then !reject absent reflections
                rej=rej+1
                write(unit=irej,fmt="(3i4,2f12.3,f10.4,f10.2)") h1(:),intens(i),
sigma(i),twtheta(i), intens(i)/sigma(i)
                cycle
            end if

            nin=nin+1 !update the number of independent reflections
            itreat(i)=i !Make this reflection treated
            sig =1.0/sigma(i)**2

```

```

ini(nin)=i      !put pointers for initial and final equivalent reflections
fin(nin)=i
nequv(nin)=1 !One reflection for the moment equivalent to itself

do j=i+1,nr      !look for equivalent reflections to the current (i)
in the list
  if(abs(twtheta(i)-twtheta(j)) > 0.001) exit
  h2=h(:,j)
  if(hkl_equiv(h1,h2,grp_espacial,Friedel)) then ! if h1 eqv h2
    itreat(j) = i ! add h2 to the
list equivalent to i
    nequv(nin)=nequv(nin)+1 ! update the
number of equivalents
    sig=sig + 1.0/sigma(j)**2
    fin(nin)=j
  end if
end do

ns=0
do j=ini(nin),fin(nin)
  if(itreat(j) == i) then
    ns=ns+1
    weight(ns)=(1.0/sigma(j)**2)/sig
  end if
end do

suma=0.0
ns=0
do j=ini(nin),fin(nin)
  if(itreat(j) == i) then
    ns=ns+1
    suma=suma+weight(ns)*intens(j)
  end if
end do

intav(nin)=suma

suma=0.0
ns=0
do j=ini(nin),fin(nin)
  if(itreat(j) == i) then
    ns=ns+1
    delt= intav(nin)-intens(j)
    if(abs(delt)/intav(nin) > warning) warn(nin)=1
    suma=suma+weight(ns)*delt*delt
  end if
end do

sigmav(nin)=sqrt(suma)
sigg=SUM(sigma(ini(nin):fin(nin)))/max(1.0,real(fin(nin)-ini(nin)))
if(sigmav(nin) < sigg) sigmav(nin) = sigg

end if !itreat
end do
!
! Second loop over reflections to calculate R-int
!
ns=0
suma =0.0
suman=0.0
do i=1,nin
  k=ini(i)
  if(nequv(i) < 2 ) cycle
  sig=0.0
  do j=ini(i),fin(i)
    if(itreat(j) == k) then
      sig=sig+1.0/sigma(j)**2
    end if
  end do
  suman=suman+sig
end do

```

```

        end if
    end do
    sig=1.0/sig
    do j=ini(i),fin(i)
        if(itreat(j) == k) then
            ns=ns+1
            suma=suma+abs(intav(i)-intens(j))
            suman=suman+intens(j)
        end if
    end do
end do
Rint = 100.0*suma/max(1.0, suman)
!
!--- Writing the list of rejected and merged reflections
!

    open(unit=ihkl, file=trim(fileout)//".int", status="replace",action="write")
    write(unit=ihkl,fmt="(a)") title
    write(unit=ihkl,fmt="(a)") "(3i4,2f12.3,i5,4f8.2)"
    write(unit=ihkl,fmt="(f9.5,a)") wavel," 0 0"
    cent=0
    do i=1,nin
        j=ini(i)
        h1(:)= h(:,j)
        if(hkl_equiv(h1,-h1,grp_espacial,.false.)) then
            cent=cent+1 !calculate the number of acentric reflections
            write(unit=ihkl,fmt="(3i4,2f12.3,i5,4f8.2,a)") h1(:),intav(i),sigmav(i),
1,0.0,0.0,0.0,0.0, warn_mess(warn(i))// " Centric"
        else
            write(unit=ihkl,fmt="(3i4,2f12.3,i5,4f8.2,a)") h1(:),intav(i),sigmav(i),
1,0.0,0.0,0.0,0.0, warn_mess(warn(i))
        end if
    end do

!----- All calculations have been done!

    write(unit=*,fmt="(/,a,i6)") " => Number of reflections read          :
", nr
    write(unit=*,fmt="(a,i6)") "  => Number of valid independent reflections:
", nin
    write(unit=*,fmt="(a,i6)") "  => Number of Centric reflections:
", cent
    write(unit=*,fmt="(a,i6)") "  => Number of rejected (absences) reflections:
", rej
    write(unit=*,fmt="(a,f6.2)") " => R-internal for equivalent reflections (%):
", Rint

    write(unit=*,fmt="(a)") "  => Program finished O.K.!, look in output
files!"
    write(unit=*,fmt="(a,a)") "          General      Output      file: ",
trim(fileout)//".out"
    write(unit=*,fmt="(a,a)") "          Independent reflections file: ",
trim(fileout)//".int"
    write(unit=*,fmt="(a,a)") "          Rejected      reflections file: ",
trim(fileout)//".rej"
    write(unit=*,fmt="(a )") "          "
    call cpu_time(t_end)
    write(unit=*,fmt="(a,f10.2,a)") " CPU-Time: ", t_end-t_start," seconds"
    stop
End Program DataRed

```

```

package reflectionSort;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;

public class Main {

    public static void main(String[] args) {
        long startTime = System.currentTimeMillis();
        ArrayList<Reflection> reflections = new ArrayList<Reflection>();
        ArrayList<SymmetryOperator> symmetryOperators = new ArrayList<SymmetryOperator>();

        // Read symmetry operators and reflections
        try {
            BufferedReader in = new BufferedReader(new FileReader("in"));
            DataFileReader reader = new DataFileReader(in);
            reader.readData(symmetryOperators, reflections);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            return;
        } catch (IOException e) {
            e.printStackTrace();
            return;
        }
        long readTime = System.currentTimeMillis();

        // Standardize reflections
        for (Reflection reflection : reflections) {
            reflection.standardizeHkl(symmetryOperators);
        }
        long standardizeTime = System.currentTimeMillis();

        // Sort reflections
        Collections.sort(reflections);
        long sortTime = System.currentTimeMillis();

        // Merge reflections, remove systematic absences, and count centric reflections
        LinkedList<Reflection> uniqueReflections = new LinkedList<Reflection>();
        List<Reflection> systematicallyAbsentReflections;
        double rint = ReflectionAnalyzer.mergeUniqueReflections(reflections, uniqueReflections, new SimpleMergeFunction());

        systematicallyAbsentReflections = ReflectionAnalyzer.removeSystematicAbsences(uniqueReflections, symmetryOperators);
        int centricReflections = ReflectionAnalyzer.countCentricReflections(uniqueReflections, symmetryOperators);

        long mergeTime = System.currentTimeMillis();

        // Print list of systematically absent reflections
        for (Reflection reflection : systematicallyAbsentReflections) {
            System.out.println(String.format("Systematically absent: %3d %3d %3d I/sigma = %5.2f", reflection.getHkl().x,
                reflection.getHkl().y, reflection.getHkl().z, reflection.getI() / reflection.getSigI()));
        }

        // Print statistics
        System.out.println(reflections.size() + " total reflections");
        System.out.println(systematicallyAbsentReflections.size() + " systematically absent reflections");
        System.out.println(uniqueReflections.size() + " unique reflections");
        System.out.println(centricReflections + " centric reflections");
        System.out.println("R(int) = " + String.format("%5.4f", rint));

        long outputTime = System.currentTimeMillis();

        boolean printTiming = true;
        if (printTiming) {
            System.out.println(String.format("%4.2f seconds reading", (readTime - startTime) / 1000.0));
            System.out.println(String.format("%4.2f seconds standardizing", (standardizeTime - readTime) / 1000.0));
            System.out.println(String.format("%4.2f seconds sorting", (sortTime - standardizeTime) / 1000.0));
            System.out.println(String.format("%4.2f seconds merging", (mergeTime - sortTime) / 1000.0));
            System.out.println(String.format("%4.2f seconds printing", (outputTime - mergeTime) / 1000.0));
            System.out.println(String.format("%4.2f seconds total", (outputTime - startTime) / 1000.0));
        }
    }
}

```



```

package reflectionSort;

import java.io.BufferedReader;
import java.io.IOException;
import java.util.List;

public class DataFileReader {

    private final BufferedReader input;

    public DataFileReader(BufferedReader in) {
        this.input = in;
    }

    public void readData(List<SymmetryOperator> symmetryOperators, List<Reflection> reflections) throws IOException {
        // Ignore first two lines of input
        String line = input.readLine();
        line = input.readLine();
        // Read line which starts with an integer indicating the number of
        // symmetry operators
        line = input.readLine();
        String[] tokens = line.split("\\s");
        int numberOfSymmetryOperators = Integer.parseInt(tokens[0]);
        // Read symmetry operators in the format:
        // r11 r12 r13 r21 r22 r23 r31 r32 r33 t1 t2 t3
        int[] matrix = new int[9];
        float[] vector = new float[3];
        for (int i = 0; i < numberOfSymmetryOperators; ++i) {
            line = input.readLine();
            tokens = line.split("\\s");
            for (int j = 0; j < matrix.length; ++j) {
                matrix[j] = Integer.parseInt(tokens[j]);
            }
            for (int j = 0; j < vector.length; ++j) {
                vector[j] = Float.parseFloat(tokens[9 + j]);
            }
            symmetryOperators.add(new SymmetryOperator(matrix, vector));
        }
        // Read read reflections in the format:
        // h k l I sigI
        line = input.readLine();
        while (line != null) {
            tokens = line.split("\\s");
            Reflection r = new Reflection(Integer.parseInt(tokens[0]), Integer.parseInt(tokens[1]), Integer
                .parseInt(tokens[2]), Double.parseDouble(tokens[3]), Double.parseDouble(tokens[4]));
            reflections.add(r);
            line = input.readLine();
        }
    }
}

```

```
package reflectionSort;
```

```
public class SymmetryOperator {

    int[] reciprocalMatrix = new int[9];
    float[] vector = new float[3];

    public SymmetryOperator(int[] matrix, float[] vector) {
        reciprocalMatrix[0] = matrix[0];
        reciprocalMatrix[1] = matrix[3];
        reciprocalMatrix[2] = matrix[6];
        reciprocalMatrix[3] = matrix[1];
        reciprocalMatrix[4] = matrix[4];
        reciprocalMatrix[5] = matrix[7];
        reciprocalMatrix[6] = matrix[2];
        reciprocalMatrix[7] = matrix[5];
        reciprocalMatrix[8] = matrix[8];
        this.vector[0] = vector[0];
        this.vector[1] = vector[1];
        this.vector[2] = vector[2];
    }

    public void applyReciprocalRotation(Tuple3i hkl, Tuple3i transformedHkl) {
        transformedHkl.x = reciprocalMatrix[0]*hkl.x + reciprocalMatrix[1]*hkl.y + reciprocalMatrix[2]*hkl.z;
        transformedHkl.y = reciprocalMatrix[3]*hkl.x + reciprocalMatrix[4]*hkl.y + reciprocalMatrix[5]*hkl.z;
        transformedHkl.z = reciprocalMatrix[6]*hkl.x + reciprocalMatrix[7]*hkl.y + reciprocalMatrix[8]*hkl.z;
    }

    public float phaseShift(Tuple3i hkl) {
        return hkl.z*vector[0] + hkl.y*vector[1] + hkl.x*vector[2];
    }

}
```

```
package reflectionSort;
```

```
import java.util.List;
```

```
class Reflection implements Comparable<Reflection>, Cloneable {
    MillerIndex hkl;

    double i;

    double sigI;

    protected Reflection() {
        hkl = new MillerIndex();
    }

    Reflection(int h, int k, int l, double i, double sigI) {
        this.hkl = new MillerIndex(h, k, l);
        this.i = i;
        this.sigI = sigI;
    }

    public Object clone() {
        Reflection r = new Reflection();
        r.hkl = new MillerIndex(hkl);
        return r;
    }

    public String toString() {
        return String.format("%4d %4d %4d %6.2f %6.2f", hkl.x, hkl.y, hkl.z, i, sigI);
    }

    void standardizeHkl(List<SymmetryOperator> symmetryOperators) {
        MillerIndex transformedHkl = new MillerIndex();
        applySymmetryOperators(symmetryOperators, transformedHkl);
    }

    private void applySymmetryOperators(List<SymmetryOperator> symmetryOperators, MillerIndex transformedHkl) {
        for (SymmetryOperator symmetryOperator : symmetryOperators) {
            symmetryOperator.applyReciprocalRotation(hkl, transformedHkl);
            setIfStandardHkl(transformedHkl);
            transformedHkl.negate();
            setIfStandardHkl(transformedHkl);
        }
    }
}
```

```

private void setIfStandardHkl(MillerIndex transformedHkl) {
    if (transformedHkl.z > hkl.z || (transformedHkl.z == hkl.z && transformedHkl.y > hkl.y)
        || (transformedHkl.z == hkl.z && transformedHkl.y == hkl.y && transformedHkl.x > hkl.x)) {
        hkl.set(transformedHkl);
    }
}

public MillerIndex getHkl() {
    return hkl;
}

public double getI() {
    return i;
}

public double getSigI() {
    return sigI;
}

@Override
public boolean equals(Object obj) {
    return hkl.equals(((Reflection) obj).hkl);
}

@Override
public int hashCode() {
    return hkl.hashCode();
}

public int compareTo(Reflection other) {
    return hkl.compareTo(other.hkl);
}

public void setI(double i) {
    this.i = i;
}

public void setSigI(double sigI) {
    this.sigI = sigI;
}
}

```

```

package reflectionSort;

```

```

import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

```

```

public class ReflectionAnalyzer {

```

```

    public static double mergeUniqueReflections(List<Reflection> reflectionsList,
        List<Reflection> uniqueReflectionsList, SimpleMergeFunction mergeFunction) {

```

```

        List<Reflection> duplicateReflections = new ArrayList<Reflection>();
        double rIntSumIDeviation = 0.0;
        double rIntSumI = 0.0;

        // Merge duplicate reflections and determine systematic absences
        PeekableIterator iter = new PeekableIterator(reflectionsList.iterator());
        do {
            Reflection reflection = (Reflection) iter.next();
            duplicateReflections.clear();
            duplicateReflections.add(reflection);
            while (iter.peek() != null && iter.peek().equals(reflection)) {
                Reflection duplicateReflection = (Reflection) iter.next();
                duplicateReflections.add(duplicateReflection);
            }
            if (duplicateReflections.size() > 1) {
                Reflection mergedReflection = mergeFunction.merge(duplicateReflections);
                for (Reflection duplicateReflection : duplicateReflections) {
                    rIntSumIDeviation += Math.abs(duplicateReflection.getI() - mergedReflection.getI());
                    rIntSumI += mergedReflection.getI();
                }
                uniqueReflectionsList.add(mergedReflection);
            } else {
                uniqueReflectionsList.add(reflection);
            }
        } while (iter.hasNext());
        return rIntSumIDeviation / rIntSumI;
    }
}

```

```

    public static List<Reflection> removeSystematicAbsences(List<Reflection> uniqueReflectionsList, List<SymmetryOperator>
        symmetryOperators) {
        ArrayList<Reflection> systematicallyAbsentReflections = new ArrayList<Reflection>();
        for (Iterator iter = uniqueReflectionsList.iterator(); iter.hasNext();) {
            Reflection reflection = (Reflection) iter.next();

```

```

        if (MillerIndex.isSystematicallyAbsent(reflection.getHkl(), symmetryOperators)) {
            iter.remove();
            systematicallyAbsentReflections.add(reflection);
        }
    }
    return systematicallyAbsentReflections;
}

public static int countCentricReflections(Collection<Reflection> uniqueReflectionsList, List<SymmetryOperator>
symmetryOperators) {
    int i = 0;
    for (Reflection reflection : uniqueReflectionsList) {
        if (MillerIndex.isCentrosymmetric(reflection.getHkl(), symmetryOperators)) {
            ++i;
        }
    }
    return i;
}
}

```

```

package reflectionSort;

import java.util.List;

class SimpleMergeFunction {

    public Reflection merge(List<Reflection> reflections) {
        double sumI = 0.0;
        double sumSigI = 0.0;
        for (Reflection reflection : reflections) {
            sumI += reflection.getI();
            sumSigI += 1.0 / (reflection.getSigI() * reflection.getSigI());
        }
        double i = sumI / reflections.size();
        double sigI = 1.0 / Math.sqrt(sumSigI);
        Reflection mergedReflection = (Reflection) reflections.get(0).clone();
        mergedReflection.setI(i);
        mergedReflection.setSigI(sigI);
        return mergedReflection;
    }
}

```

```

package reflectionSort;

import java.util.List;

public class MillerIndex extends Tuple3i {

    public MillerIndex() {
        super();
    }

    public MillerIndex(MillerIndex hkl) {
        super(hkl);
    }

    public MillerIndex(int h, int k, int l) {
        super(h, k, l);
    }

    void standardize(List<SymmetryOperator> symmetryOperators) {
        MillerIndex transformedHkl = new MillerIndex();
        for (SymmetryOperator symmetryOperator : symmetryOperators) {
            symmetryOperator.applyReciprocalRotation(this, transformedHkl);
            setIfStandard(transformedHkl);
            transformedHkl.negate();
            setIfStandard(transformedHkl);
        }
    }

    private void setIfStandard(MillerIndex transformedHkl) {
        if (transformedHkl.z > z || (transformedHkl.z == z && transformedHkl.y > y)
            || (transformedHkl.z == z && transformedHkl.y == y && transformedHkl.x > x)) {
            set(transformedHkl);
        }
    }

    public static boolean isSystematicallyAbsent(MillerIndex hkl, List<SymmetryOperator> symmetryOperators) {
        for (SymmetryOperator symmetryOperator : symmetryOperators) {
            if (isSymmetric(hkl, symmetryOperator)) {
                double n = symmetryOperator.phaseShift(hkl);
                if (Math.abs(Math rint(n) - n) > 0.001) {
                    return true;
                }
            }
        }
        return false;
    }

    public static boolean isCentrosymmetric(MillerIndex hkl, List<SymmetryOperator> symmetryOperators) {
        for (SymmetryOperator symmetryOperator : symmetryOperators) {
            if (isCentrosymmetric(hkl, symmetryOperator)) {
                return true;
            }
        }
        return false;
    }

    static boolean isSymmetric(MillerIndex hkl, SymmetryOperator symmetryOperator) {
        MillerIndex transformedHkl = new MillerIndex();
        symmetryOperator.applyReciprocalRotation(hkl, transformedHkl);
        return hkl.equals(transformedHkl);
    }

    static boolean isCentrosymmetric(MillerIndex hkl, SymmetryOperator symmetryOperator) {
        MillerIndex transformedHkl = new MillerIndex();
        symmetryOperator.applyReciprocalRotation(hkl, transformedHkl);
        transformedHkl.negate();
        return transformedHkl.equals(hkl);
    }
}

```

```

package reflectionSort;

public class Tuple3i implements Comparable<Tuple3i> {

    public int x;
    public int y;
    public int z;

    public Tuple3i() {
    }

    public Tuple3i(Tuple3i t) {
        this(t.x, t.y, t.z);
    }

    public Tuple3i(int x, int y, int z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public Tuple3i(int[] xyz) {
        this(xyz[0], xyz[1], xyz[2]);
    }

    public String toString() {
        return String.format("%d %d %d", x, y, z);
    }

    public int hashCode() {
        int result = x;
        result += 29 * y;
        result += 29 * z;
        return result;
    }

    public boolean equals(Object o) {
        return compareTo((MillerIndex) o) == 0;
    }

    public int compareTo(Tuple3i o) {
        int result = z - o.z;
        if (result == 0) {
            result = y - o.y;
            if (result == 0) {
                result = x - o.x;
            }
        }
        return result;
    }

    public void negate() {
        x *= -1;
        y *= -1;
        z *= -1;
    }

    public void set(Tuple3i t) {
        set(t.x, t.y, t.z);
    }

    public void set(int x, int y, int z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }
}

```

```

package reflectionSort;

import java.util.Iterator;

public class PeekableIterator implements Iterator {

    Object next;
    Object peek;
    Iterator iter;

    PeekableIterator(Iterator iter) {
        this.iter = iter;
        if (iter.hasNext()) {
            peek = iter.next();
        }
    }

    public boolean hasNext() {
        return peek != null;
    }

    public Object next() {
        next = peek;
        if (iter.hasNext()) {
            peek = iter.next();
        } else {
            peek = null;
        }
        return next;
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }

    public Object peek() {
        return peek;
    }
}

```

```

program sortmerge

implicit none

integer::n_reflection,n_rejected,n_unique,n_centric,n_new_unique
real::rint,fob,sf,temp
integer::rots_symm(3,3,6)
real::tran_symm(3,6)
integer::ind_h,ind_k,ind_l,temp_h,temp_k,temp_l
integer::index_h(400000),index_k(400000),index_l(400000)
real::fobs(400000),sigmaf(400000)
integer::flag,centric,new_h,new_k,new_l,n_symm,centric_flag

integer::a,b

C-----C
C-- Read reflection file                                --C
C-----C

centric = 0
n_reflection = 1
n_rejected = 0
rint = 0
n_unique = 0
n_symm=6
n_centric = 0

open(1,file='in')
read(1,*)
read(1,*)
read(1,*)

do a = 1 , n_symm
  read(1,*)rots_symm(1,1,a),rots_symm(1,2,a),rots_symm(1,3,a),
+      rots_symm(2,1,a),rots_symm(2,2,a),rots_symm(2,3,a),
+      rots_symm(3,1,a),rots_symm(3,2,a),rots_symm(3,3,a),
+      tran_symm(1,a),tran_symm(2,a),tran_symm(3,a)
end do

do
  read ( 1 , * , end = 999 ) ind_h , ind_k , ind_l , fob , sf

C-----C
C-- Reorder indices to a standard form:                --C
C--                                                    --C
C-- l positive, l=0 => k positive, l=k=0 => h positive --C
C-----C

new_h = ind_h
new_k = ind_k
new_l = ind_l

do a = 1 , n_symm

  temp_h = ind_h * rots_symm(1,1,a) + ind_k * rots_symm(2,1,a)
+      + ind_l * rots_symm(3,1,a)
  temp_k = ind_h * rots_symm(1,2,a) + ind_k * rots_symm(2,2,a)
+      + ind_l * rots_symm(3,2,a)
  temp_l = ind_h * rots_symm(1,3,a) + ind_k * rots_symm(2,3,a)
+      + ind_l * rots_symm(3,3,a)

  if ( (temp_l .LT. 0) .OR. (temp_l .EQ. 0 .AND. temp_k .LT. 0)
+      .OR. (temp_l .EQ. 0 .AND. temp_k .EQ. 0 .AND. temp_h
+      .LT. 0) ) then
    temp_h = -temp_h
    temp_k = -temp_k
    temp_l = -temp_l
  end if
end do

```



```

        end if

        if ( (temp_l .GT. new_l) .OR. (temp_l .EQ. new_l .AND. temp_k
+         .GT. new_k) .OR. (temp_l .EQ. new_l .AND. temp_k .EQ. new_k
+         .AND. temp_h .GT. new_h)) then
            new_h = temp_h
            new_k = temp_k
            new_l = temp_l
        end if
    end do

    index_h(n_reflection) = new_h
    index_k(n_reflection) = new_k
    index_l(n_reflection) = new_l

    fobs(n_reflection) = fob
    sigmaf(n_reflection) = sf

    n_reflection = n_reflection + 1

end do

999  continue

    n_reflection = n_reflection - 1

C-----C
C-- Processing the data.                                --C
C-----C

    call sort_hkl(n_reflection,index_l,index_k,index_h,fobs,sigmaf)

    call calculate_rint_value(n_reflection,index_h,index_k,index_l,
+         fobs,sigmaf,rint)

    call merge_equivalent_reflections(n_reflection,index_h,index_k,
+         index_l,fobs,sigmaf,n_unique)

C-----C
C-- Identify a reflection, that should be systematically absent and count --C
C-- the number of centric reflections.                                --C
C-----C

    n_new_unique = 0
    centric_flag = 0
    outer: do b = 1 , n_unique
        do a = 2 , n_symm

            temp_h = index_h(b) * rots_symm(1,1,a)
+         + index_k(b) * rots_symm(2,1,a)
+         + index_l(b) * rots_symm(3,1,a)
            temp_k = index_h(b) * rots_symm(1,2,a)
+         + index_k(b) * rots_symm(2,2,a)
+         + index_l(b) * rots_symm(3,2,a)
            temp_l = index_h(b) * rots_symm(1,3,a)
+         + index_k(b) * rots_symm(2,3,a)
+         + index_l(b) * rots_symm(3,3,a)

            ! Look for systematically absent reflections

            if ( temp_h .EQ. index_h(b) .AND. temp_k .EQ. index_k(b) .AND.
+             temp_l .EQ. index_l(b) ) then
                temp = index_h(b) * tran_symm(1,a)
+             + index_k(b) * tran_symm(2,a)
+             + index_l(b) * tran_symm(3,a) + 999.5
                if ( temp - int(temp) .LT. 0.4 ) then
                    n_rejected = n_rejected + 1
                end if
            end if
        end do
    end do

```

```

      print *, 'Reflection ', index_h(b), index_k(b), index_l(b)
+      , ' is systematically absent with I/sigma = '
+      , fobs(b)/sigmaf(b)
      centric_flag = 0
      cycle outer
    end if

    ! flag centric reflections

    elseif ( temp_h .EQ. -index_h(b) .AND.
+           temp_k .EQ. -index_k(b) .AND.
+           temp_l .EQ. -index_l(b) ) then
      centric_flag = 1
    end if

    end do
    n_centric = n_centric + centric_flag
    centric_flag = 0
    n_new_unique = n_new_unique + 1
    index_h(n_new_unique) = index_h(b)
    index_k(n_new_unique) = index_k(b)
    index_l(n_new_unique) = index_l(b)
    fobs(n_new_unique) = fobs(b)
    sigmaf(n_new_unique) = sigmaf(b)
  end do outer

  print *, n_reflection, ' Reflections were reduced to ', n_new_unique
+  , ' Reflections '
  print *, 'of which ', n_centric, ' were centric.'
  print *, 'R(int)=', rint
  print *, n_rejected, ' Reflections were systematically absent.'

  close(1)

end

```

```

C=====C
C
C Soubroutine for sorting hkl-indices
C
C This routine sorts the hkl indices in the following way:
C fastest changing index: index_3
C slowest changing index: index_1
C
C=====C

```

```

      subroutine sort_hkl(n_reflection, index_1, index_2, index_3,
+      fobs, phi)

      implicit none

      integer::n_reflection
      integer::index_1(n_reflection), index_2(n_reflection),
+      index_3(n_reflection)
      real::fobs(n_reflection), phi(n_reflection)

```

```

C-----C
C-- Local variables
C-----C

```

```

      integer::t1, t2, position, start
      integer, allocatable::n_index_1(:), n_index_2(:), n_index_3(:)
      integer, allocatable::pointer_1(:), pointer_2(:),
+      pointer_3(:, :), pointer_4(:, :)
      integer::max_1, min_1, max_2, min_2, max_3, min_3

```

```

integer::index_1_temp(n_reflection),index_2_temp(n_reflection),
+      index_3_temp(n_reflection)
real::fobs_temp(n_reflection),phi_temp(n_reflection)

integer::i,j,k

C-----C
C-- Sort on index_1                                --C
C-----C

C-----C
C-- Determine minimum and maximum index in index_1    --C
C-----C

max_1 = -500
min_1 = 500
max_2 = -500
min_2 = 500

do i = 1 , n_reflection
  if ( index_1(i) .GT. max_1 ) then
    max_1 = index_1(i)
  end if
  if ( index_1(i) .LT. min_1 ) then
    min_1 = index_1(i)
  end if
  if ( index_2(i) .GT. max_2 ) then
    max_2 = index_2(i)
  end if
  if ( index_2(i) .LT. min_2 ) then
    min_2 = index_2(i)
  end if
end do

allocate (n_index_1(min_1:max_1))

do i = min_1 , max_1 + 1
  n_index_1(i) = 0
end do

C-----C
C-- Scan list and count how often each index occurs    --C
C-----C

do i = 1 , n_reflection
  t1 = index_1(i)
  n_index_1(t1) = n_index_1(t1) + 1
  index_1_temp(i) = index_1(i)
  index_2_temp(i) = index_2(i)
  index_3_temp(i) = index_3(i)
  fobs_temp(i) = fobs(i)
  phi_temp(i) = phi(i)
end do

C-----C
C-- Define an array with pointers to the positions in the sorted list    --C
C-----C

allocate (pointer_1(min_1:max_1))
allocate (pointer_2(min_1:max_1))
allocate (pointer_3(min_1:max_1,min_2:max_2))
allocate (pointer_4(min_1:max_1,min_2:max_2))

t1 = 0

do i = min_1 , max_1
  t2 = n_index_1(i)

```

```

      n_index_1(i) = t1
      pointer_1(i) = t1 + 1
      t1 = t1 + t2
      pointer_2(i) = t1
    end do

```

```

C-----C
C-- Scan list again and put each index at the final position      --C
C-----C

```

```

    do i = 1 , n_reflection
      t1 = index_1_temp(i)
      position = n_index_1(t1) + 1
      index_1(position) = index_1_temp(i)
      index_2(position) = index_2_temp(i)
      index_3(position) = index_3_temp(i)
      fobs(position) = fobs_temp(i)
      phi(position) = phi_temp(i)
      n_index_1(t1) = position
    end do

```

```

    deallocate (n_index_1)

```

```

C-----C
C-- Sort on index_2      --C
C-----C

```

```

    start = 0

```

```

    do i = min_1 , max_1

```

```

      max_2 = -500
      min_2 = 500

```

```

      do j = pointer_1(i) , pointer_2(i)
        if ( index_2(j) .GT. max_2 ) then
          max_2 = index_2(j)
        end if
        if ( index_2(j) .LT. min_2 ) then
          min_2 = index_2(j)
        end if
      end do

```

```

      allocate (n_index_2(min_2:max_2))

```

```

      do j = min_2 , max_2 + 1
        n_index_2(j) = 0
      end do

```

```

      do j = pointer_1(i) , pointer_2(i)
        t1 = index_2(j)
        n_index_2(t1) = n_index_2(t1)+1
        index_1_temp(j) = index_1(j)
        index_2_temp(j) = index_2(j)
        index_3_temp(j) = index_3(j)
        fobs_temp(j) = fobs(j)
        phi_temp(j) = phi(j)
      end do

```

```

      t1 = start

```

```

      do j = min_2 , max_2
        t2 = n_index_2(j)
        n_index_2(j) = t1
        pointer_3(i,j) = t1 + 1
        t1 = t1 + t2
        pointer_4(i,j) = t1
      end do

```

```

end do

start = pointer_4(i,max_2)

do j = pointer_1(i) , pointer_2(i)
  t1 = index_2_temp(j)
  position = n_index_2(t1) + 1
  index_1(position) = index_1_temp(j)
  index_2(position) = index_2_temp(j)
  index_3(position) = index_3_temp(j)
  fobs(position) = fobs_temp(j)
  phi(position) = phi_temp(j)
  n_index_2(t1) = position
end do

deallocate (n_index_2)

end do

C-----C
C-- Sort on index_3 --C
C-----C

start = 0

do i = min_1 , max_1

  max_2 = -500
  min_2 = 500

  do j = pointer_1(i) , pointer_2(i)
    if ( index_2(j) .GT. max_2 ) then
      max_2 = index_2(j)
    end if
    if ( index_2(j) .LT. min_2 ) then
      min_2 = index_2(j)
    end if
  end do

  do j = min_2 , max_2

    max_3 = -500
    min_3 = 500

    do k = pointer_3(i,j) , pointer_4(i,j)
      if ( index_3(k) .GT. max_3 ) then
        max_3 = index_3(k)
      end if
      if ( index_3(k) .LT. min_3 ) then
        min_3 = index_3(k)
      end if
    end do

    allocate (n_index_3(min_3:max_3))

    do k = min_3 , max_3 + 1
      n_index_3(k) = 0
    end do

    do k = pointer_3(i,j) , pointer_4(i,j)
      t1 = index_3(k)
      n_index_3(t1) = n_index_3(t1) + 1
      index_1_temp(k) = index_1(k)
      index_2_temp(k) = index_2(k)
      index_3_temp(k) = index_3(k)
      fobs_temp(k) = fobs(k)
      phi_temp(k) = phi(k)
    end do
  end do
end do

```

```

    end do

    t1 = start

    do k = min_3,max_3
        t2 = n_index_3(k)
        n_index_3(k) = t1
        t1 = t1 + t2
    end do

    start = t1

    do k = pointer_3(i,j) , pointer_4(i,j)
        t1 = index_3_temp(k)
        position = n_index_3(t1) + 1
        index_1(position) = index_1_temp(k)
        index_2(position) = index_2_temp(k)
        index_3(position) = index_3_temp(k)
        fobs(position) = fobs_temp(k)
        phi(position) = phi_temp(k)
        n_index_3(t1) = position
    end do

    deallocate (n_index_3)

end do

end do

deallocate (pointer_1,pointer_2,pointer_3,pointer_4)

return
end

C=====C
C                                                    C
C Subroutine for merging equivalent reflections.      C
C                                                    C
C=====C

    subroutine merge_equivalent_reflections(nref,inh,ink,inl,fob,
+                                          sig,uni)

    implicit none

    integer::nref,inh(nref),ink(nref),inl(nref)
    integer::uni
    real::fob(nref),sig(nref)

    real::fmean,sum_s,sum_1,fobs_new,sig_new

    integer::i,j,k

    i = 1
    j = 1
    uni = 0
    fmean = fob(i)
    sum_s = 1/sig(i)**2

    do
        if (inh(i) .EQ. inh(i+j) .AND. ink(i) .EQ. ink(i+j) .AND.
+          inl(i) .EQ. inl(i+j) ) then
            fmean = fmean + fob(i+j)
            sum_s = sum_s + 1/sig(i+j)**2
            j = j + 1
            if ( ( i + j ) .LE. nref ) then

```

```

        cycle
      end if
    end if
    fobs_new = fmean / j
    sig_new = 1/sqrt(sum_s)
    uni = uni + 1
    inh(uni) = inh(i)
    ink(uni) = ink(i)
    inl(uni) = inl(i)
    fob(uni) = fobs_new
    sig(uni) = sig_new

    i = i + j
    j = 1
    if ( i + j .LE. nref ) then
      fmean = fob(i)
      sum_s = 1/sig(i)**2
      fobs_new = 0
      sig_new = 0
    else
      uni = uni + 1
      inh(uni) = inh(i)
      ink(uni) = ink(i)
      inl(uni) = inl(i)
      fob(uni) = fmean
      sig(uni) = 1/sqrt(sum_s)
      exit
    end if
  end do

  return
end

```

```

C=====C
C
C Subroutine for calculating Rint values.
C
C=====C

```

```

subroutine calculate_rint_value(uni,indh,indk,indl,fob,sig,ri)

implicit none

integer::uni,indh(uni),indk(uni),indl(uni)
real::fob(uni),sig(uni),ri

real::sum_i,temp1,temp2

integer::a,b,c

temp1 = 0
temp2 = 0
a = 1
do while ( a < uni + 1 )
  b = 1
  sum_i = fob(a)
  do
    if (indh(a) .EQ. indh(a+b) .AND. indk(a) .EQ. indk(a+b) .AND.
+     indl(a) .EQ. indl(a+b) ) then
      sum_i = sum_i + fob(a+b)
      b = b + 1
      if (a+b .LE. uni) then
        cycle
      end if
    end if
    if (b.NE.1) then
      sum_i = sum_i / b
    end if
  end do
  a = a + 1
end do

```

```
        do c = a , a + b - 1
            temp1 = temp1 + abs(fob(c) - sum_i)
            temp2 = temp2 + fob(c)
        end do
    end if
    a = a + b
    b = 1
    exit
end do
end do

ri = temp1 / temp2

return
end
```



```

from cctbx.array_family import flex
from cctbx import crystal
from cctbx import uctbx
from cctbx import sgtbx
from cctbx import miller
import sys

def run(args):
    assert len(args) == 1
    lines = open(args[0]).read().splitlines()
    title = lines[0]
    unit_cell = uctbx.unit_cell(lines[1])
    n_symops = int(lines[2].split()[0])
    space_group = sgtbx.space_group()
    for line in lines[3:3+n_symops]:
        coeffs = [float(field) for field in line.split()]
        space_group.expand_smx(sgtbx.rt_mx(coeffs[:9], coeffs[9:]))
    crystal_symmetry = crystal.symmetry(
        unit_cell=unit_cell,
        space_group=space_group)
    miller_indices = flex.miller_index()
    data = flex.double()
    sigmas = flex.double()
    for i_line in xrange(3+n_symops, len(lines)):
        fields = lines[i_line].split()
        assert len(fields) == 5
        miller_indices.append([int(value) for value in fields[:3]])
        data.append(float(fields[3]))
        sigmas.append(float(fields[4]))
    miller_set=miller.set(
        crystal_symmetry=crystal_symmetry,
        indices=miller_indices,
        anomalous_flag=False)
    miller_array = miller_set.array(
        data=data,
        sigmas=sigmas).set_observation_type_xray_intensity()
    print "Before merging:"
    miller_array.show_summary()
    print
    merged = miller_array.merge_equivalents().array().sort(by_value="data")
    print "After merging:"
    merged.show_comprehensive_summary().show_array()
    print

if (__name__ == "__main__"):
    run(sys.argv[1:])

```

```

#
# cctbx sort-merge solution for Siena exercise given by George Sheldrick
#
# sort_merge_initial.py was written in exactly 30 minutes while
# sitting in the audience as others explained their solutions.
#
# sort_merge.py is a slight enhancement (use diff for details).
# It doesn't solve the exercise exactly, but demonstrates how to
# work with the high-level cctbx facilities to solve most of the
# exercise. Note that sort_merge.py produces significantly more
# information than was requested, e.g. the space group name,
# data completeness, etc.
#

from cctbx.array_family import flex
from cctbx import crystal
from cctbx import uctbx
from cctbx import sgtbx
from cctbx import miller
import sys

def run(args):
    assert len(args) == 1
    lines = open(args[0]).read().splitlines()
    title = lines[0]
    unit_cell = uctbx.unit_cell(lines[1])
    n_symops = int(lines[2].split()[0])
    space_group = sgtbx.space_group()
    for line in lines[3:3+n_symops]:
        coeffs = [float(field) for field in line.split()]
        space_group.expand_smx(sgtbx.rt_mx(coeffs[:9], coeffs[9:]))
    crystal_symmetry = crystal.symmetry(
        unit_cell=unit_cell,
        space_group=space_group)
    miller_indices = flex.miller_index()
    data = flex.double()
    sigmas = flex.double()
    for i_line in xrange(3+n_symops, len(lines)):
        fields = lines[i_line].split()
        assert len(fields) == 5
        miller_indices.append([int(value) for value in fields[:3]])
        data.append(float(fields[3]))
        sigmas.append(float(fields[4]))
    miller_set=miller.set(
        crystal_symmetry=crystal_symmetry,
        indices=miller_indices,
        anomalous_flag=False)
    miller_array = miller_set.array(
        data=data,
        sigmas=sigmas).set_observation_type_xray_intensity()
    print "Before merging:"
    miller_array.show_summary()
    print
    merged = miller_array.merge_equivalents()
    merged.show_summary()
    print
    merged_array = merged.array()
    print "After merging:"
    merged_array.show_comprehensive_summary()
    print

if (__name__ == "__main__"):
    run(sys.argv[1:])

```

# CrysFML: A crystallographic library in modern Fortran

Juan Rodríguez-Carvajal  
Laboratoire Léon Brillouin, (CEA-CNRS), CEA/ Saclay  
FRANCE

## Content of the talk

- ➔ Scientific Computing: Why Fortran?
- ➔ Crystallographic computing: CrysFML



## Programming paradigms for scientific applications (I)

- ➔ Procedural, imperative structured programming (PP)  
Pascal, C, Fortran 77, ...
- ➔ Module-Oriented Programming (MOP)  
Fortran 95, ADA95, Modula-2, ... **Fortran 2003**
- ➔ Object oriented programming (OOP)  
C++, Java, Smalltalk, Eiffel, ADA95, ...  
**Fortran 2003**



## Programming paradigms for scientific applications (II). Why Fortran?

### Some reasons for developing in modern Fortran

- ➔ Simplicity and clarity of the syntax and the new facilities for global array manipulation. This is important for the common scientist that may write programs occasionally. This makes programming in **Fortran** more natural and problem solving oriented.
- ➔ Availability of many OOP features in modern **Fortran**: user-defined types, encapsulation, overload of procedures and functions. The lacking features (e.g. direct inheritance and class methods) are of less importance for scientific computing than those already available (all of them are available in **Fortran 2003**).



## Programming paradigms for scientific applications (III). Why Fortran?

### Some reasons for developing in modern Fortran

- ➔ The powerful implicit interface provided by encapsulating all functions and subroutines in *modules*, allowing to catch many errors at compile time, if one uses the *intent* attribute for procedure arguments. *We may consider that Module Oriented Programming as an alternative/complement to OOP.*
- ➔ Efficiency of the generated executable codes compared to C/C++ programs of similar complexity.
- ➔ Compatibility with legacy code and availability of a huge amount of free mathematical subroutines and functions. Re-usability of procedures written in **Fortran 77** was already a reality.



## Programming paradigms for scientific applications (IV) . Why Fortran?

### Some reasons for developing in modern Fortran

- ➔ The new standard (published in November 2004): **Fortran 2003** contains all necessary features to perform pure OOP
- ➔ John Reid, WG5 Convener: *The new features of Fortran 2003*, PDF document available directly from the Internet:  
<http://ftp.nag.co.uk/sc22wg5/N1551-N1600/N1579.pdf>
- ➔ To our knowledge **Fortran 2003** exist partially in NagF95, G95, in the new Lahey compiler for .NET, ...



## Existing Crystallographic Libraries (CCSL)

### CCSL (Crystallographic Cambridge Subroutines Library)

J. Brown, J.C. Matthewmann  
(W.I.F. David for powder diffraction)

⇒ The most complete set of procedures for crystallographic calculations. Well documented.

⇒ Written in Fortran 77 and with single crystal work in mind. Profuse use of commons. Difficult to adapt to modern programming techniques.

## Existing Crystallographic Libraries (cctbx, Clipper)

### Computational Crystallography Toolbox (cctbx)

R.W. Grosse-Kunstleve, P.D. Adams....

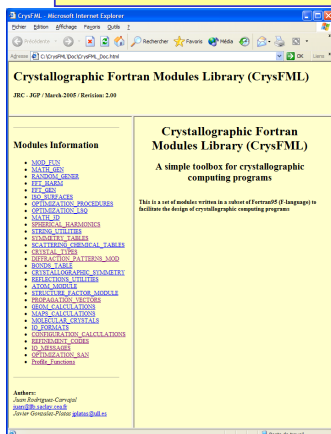
### Clipper

Kevin Cowtan

⇒ Written in C++ and handled using Python scripts.



### CrysFML: a collection of F-modules for crystallography

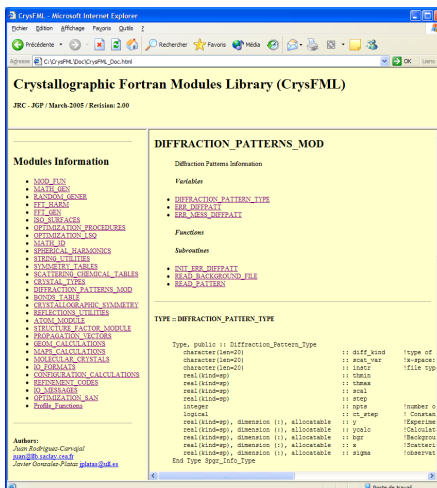


*"Crystallographic Fortran Modules Library (CrysFML). A simple toolbox for crystallographic computing programs"*

*Commission on Crystallographic Computing, IUCr Newsletter No.1, pp 50-58, January 2003.*

There are many other modules that are not ready for distribution: Magnetism,

Cost functions  
Instrument descriptions (Four circles + large PSD)  
Refinement codes for molecular crystals



### CrysFML info

In some cases the information about a particular procedure doesn't appear, then goto the source code!

The reason: I didn't obey my own rules for documentation!



## Developers of CrysFML/WinPLOTR/FullProf

Juan Rodríguez-Carvajal (LLB, France)

CrysFML, FullProf, Baslreps, Simbo, Enemag, Polar3D,...

Javier González-Platas (ULL, Tenerife, Spain)

CrysFML, GUIs, GFourier, EdPCR

⇒ Contributors:

⇒ Thierry Roisnel (LCSIM, Rennes, France)

WinPLOTR

⇒ Carlos Frontera (ICMAB, Barcelona, Spain)

Polarized neutrons, Flipping ratio data handling

⇒ Marc Janoschek (PSI, Villigen, Switzerland)

Polarized neutrons, 3D-Polarimetry

⇒ Laurent Chapon & Aziz Daoud-Aladine (ISIS, U.K.)

T.O.F. powder diffraction, WCrysfGL, Fp\_Studio

Incommensurate crystal structures



## Developers of ... We are not professional programmers!

⇒ Juan Rodríguez-Carvajal (LLB, CEA-CNRS, France)

Structural, electronic and magnetic properties of oxides and intermetallics. Modeling of magnetic structures

⇒ Javier González-Platas (ULL, Tenerife, Spain)

Crystal structure determination of organic natural compounds.

Teaching in Physics.

⇒ Thierry Roisnel (LCSIM, Rennes, France)

Crystal structure determination of cluster compounds

Single Crystal X-ray diffraction service (U. of Rennes)

⇒ Carlos Frontera (ICMAB, Barcelona, Spain)

Magnetic properties of oxides

⇒ Aziz Daoud-Aladine (ISIS, UK)

Charge, spin and orbital ordering in manganites (Co-resp. SXD)

⇒ Laurent Chapon (ISIS, UK)

Thermo-electrics, multi-ferroics, ... (Co-resp. GEM)

⇒ Marc Janoschek (PSI, Villigen, Switzerland)

Polarized neutrons instrumentation, Mu-PAD



## Scope of CrysFML

We have developed a set of **Fortran 95 modules**, Crystallographic Fortran Modules Library (**CrysFML**), that may be **used** (in the Fortran 95 sense) in crystallographic and diffraction computing programs.

⇒ Modern **array syntax and new features of Fortran 95** are **used** through the modules. In fact the whole library is written in **F-language**, a strict subset of Fortran 95 for which free compilers are available.

⇒ We take advantage of all **object oriented programming** (OOP) techniques already available in Fortran: **user-defined types**, **encapsulation**, **overload (polymorphism) of procedures and functions**. The lacking features (e.g. inheritance and class methods) will be easily implemented as soon as Fortran 2003 compilers become available.

⇒ **Main programs** using the adequate modules may perform more or less complicated calculations with only **few lines of code**.



## F-language (strict subset of Fortran 95)

All free F-compilers can be downloaded from the site:

***ftp://ftp.swcp.com/~walt/pub/F***

See also:

***http://www.fortran.com/fortran/Imagine1***



## Free Fortran 95 compiler G95: strong development

All implementations of the G95-compiler (based in gcc) can be downloaded from the G95 home page:

***http://www.g95.org***

**Platforms: Linux, Windows, Mac OS, Solaris, OpenBSD, etc...**



## Present status of CrysFML

⇒ The present **CrysFML** contains general and specific **Mathematical modules** (FFTs, geometrical calculations, optimizers, matrix operations). Procedures for reading files of many different formats, **string utilities** for handling free format, **generation and reading of CIF files**.

⇒ Modules for generating space groups from their **Hermann-Mauguin** or **Hall symbols**. Generic space groups with non-conventional lattice centring vectors can also be built using **user-defined generators**.

⇒ Reflection handling modules, including propagation vectors, may be used for generating reflections in selected regions of reciprocal space and for calculating structure factors.

⇒ The **documentation is written within the source code** using special comment symbols. A **document**, in **HTML format**, containing the description of all modules and procedures **can be generated** using a Fortran program (get\_doc).



## Present status of CrysFML

⇒ At present there is no formal way of distributing **CrysFML**, I can send copies (of the most stable modules) by e-mail to everyone wishing to use it.

⇒ There are parts of the library that are not completely developed so be patient and comprehensive.

⇒ The library is distributed with a set of working examples so that the user can mimic in order to create his (her) own programs.



## Programs using CrysFML (I)

**FullProf** : Crystal and magnetic structure refinement, powder/single crystals, polarised neutrons, constant wavelength, TOF, energy dispersive, multiple patterns.

**FOURIER, GFOURIER and EdPCR**. These programs work on Windows and Linux and are already distributed from the LLB Web site.

**BasIREPS**: Program for calculating basis functions of irreducible representations of space groups. This program is useful for determining magnetic structures and phonon symmetry analysis.

**SIMBO**: Program for the analysis of the magnetic topology of an arbitrary crystal structure. Generates a formal description of the Fourier transform of the exchange interactions to be used by other programs.

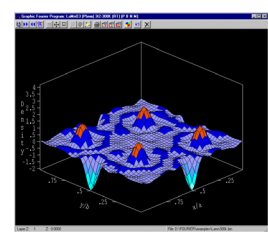
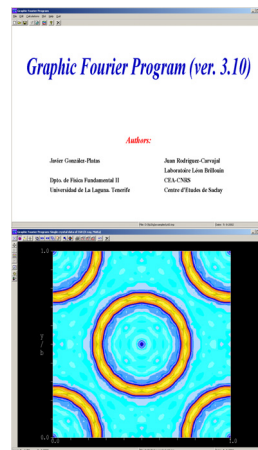


## Programs using CrysFML (II)

**ENERMAG:** Program to analyse the classical magnetic energy as a function of the exchange interactions and the point in the Brillouin Zone. This program can be used to generate theoretical magnetic phase diagrams in the J-space in order to get insight into the experimentally determined magnetic structures.

**SIMILAR:** Program to make conversion of settings for describing crystallographic structures. It determines automatically the splitting of Wyckoff positions on going from a space group to one of their subgroups. Calculate all the *translationengleiche* subgroups of a space group, co-set decompositions, etc.

**DATARED:** Program for data reduction of single crystal data. It handles twinning and incommensurate magnetic and crystal structures. Prepares files to be read by **FullProf** when using single crystals.

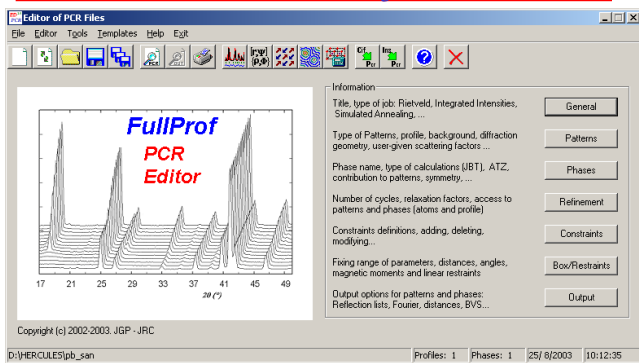


The programs **Gfourier** and **Fourier** are based in **CrysFML**

Graphic utilities: **Winteracter**  
<http://www.winteracter.com>



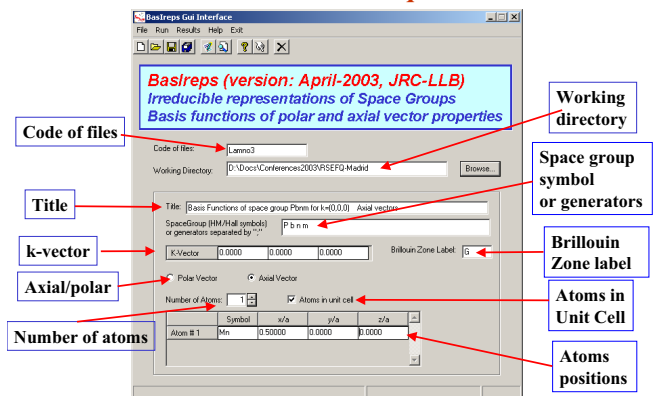
## A GUI for *FullProf*: EdPCR



GUI using Winteracter: <http://www.winteracter.com>



## GUI for *BasIreps*



<ftp://ftp.cea.fr/pub/lhb/divers/fullprof.2k>



## Example of *BasIreps* output: \*.bsr

```

PROPAGATION VECTOR GROUP INFORMATION
=====
=> The input propagation vector is: K=( 0.5000 0.5000 0.5000 )
=> K .. IS NOT .. equivalent to -K
=> The operators following the k-vectors constitute the co-set decomposition G[Gk]
The list of equivalent k-vectors are also given on the set of operators.
=> The star of K is formed by the following 2 vectors:

k_1 = ( 0.5000 0.5000 0.5000 )
Op: ( 1 ) x,y,z
Op: ( 3 ) x,-y,-z -> ( 0.5000 -0.5000 -0.5000 )
Op: ( 4 ) -x+1/2,-y,-z+1/2 -> ( -0.5000 -0.5000 0.5000 )
Op: ( 7 ) -x+1/2,y,-z+1/2 -> ( -0.5000 0.5000 -0.5000 )
Op: ( 10 ) y+3/4,-x+1/4,-z+3/4 -> ( -0.5000 0.5000 -0.5000 )
Op: ( 13 ) -y+3/4,-x+1/4,z+3/4 -> ( -0.5000 -0.5000 0.5000 )
Op: ( 14 ) -y+3/4,x+3/4,-z+1/4 -> ( 0.5000 -0.5000 -0.5000 )
Op: ( 16 ) y+3/4,x+3/4,z+1/4 -> ( 0.5000 0.5000 0.5000 )

Equiv. -k: k_2 = ( 0.5000 -0.5000 0.5000 )
Op: ( 2 ) -y+1/4,x+3/4,z+1/4 -> ( 0.5000 0.5000 -0.5000 )
Op: ( 5 ) y+1/4,x+3/4,-z+1/4 -> ( -0.5000 0.5000 0.5000 )
Op: ( 6 ) y+1/4,-x+1/4,z+3/4 -> ( -0.5000 -0.5000 -0.5000 )
Op: ( 8 ) -y+1/4,-x+1/4,-z+3/4 -> ( -0.5000 -0.5000 -0.5000 )
Op: ( 9 ) -x,-y,-z -> ( -0.5000 -0.5000 -0.5000 )
Op: ( 11 ) -x,y,z -> ( -0.5000 0.5000 0.5000 )
Op: ( 12 ) x+1/2,y,-z+1/2 -> ( 0.5000 0.5000 -0.5000 )
Op: ( 15 ) x+1/2,-y,-z+1/2 -> ( 0.5000 -0.5000 0.5000 )

=> G_k has the following symmetry operators:
1 SYMM( 1 ) = x,y,z
2 SYMM( 3 ) = x,-y,-z
3 SYMM( 4 ) = -x+1/2,-y,-z+1/2
4 SYMM( 7 ) = -x+1/2,y,-z+1/2

```



## Example of *BasIreps* output: \*.bsr

```

=> Number of elements of G_k: 8
=> Number of irreducible representations of G_k: 2
=> Dimensions: 2 2

=> Symmetry elements of G_k and ireps:
Symmetry elements reduced to the standard form (positive translations < 1)
The matrices of IRreps have been multiplied by the appropriate phase factor

-> SYMM_K( 2 ): -x+1/2,-y,-z+1/2 : 2 ( 0, 0, z ) -> h4
Phase factor for correcting input data: 0.0000
Matrix of IRrep( 1 ):
i 0
0 -1

Matrix of IRrep( 2 ):
i 0
0 -1

-> SYMM_K( 8 ): y+3/4,x+3/4,z+1/4 : m ( x, x, z ) -> h37
Phase factor for correcting input data: 1.5000
Matrix of IRrep( 1 ):
0 i
-1 0

Matrix of IRrep( 2 ):
0 -i
1 0

```



## Example of *BasIreps* output: \*.bsr

+++++ Basis functions of Representation IRrep(1) of dimension 2 contained 3 times in GAMMA +++++

SYMM x,y,z -x+1/2,-y,z-1/2 y+3/4,-x+1/4,-z+3/4 -y+1/4,x+1/4,-z+3/4

```
Atoms: Cu_1 Cu_2 Cu_3 Cu_4
1:Re ( 1 0 0) ( 0 0 0) ( 0 -1 0) ( 0 -1 0)
1m ( 0 0 0) ( -1 0 0) ( 0 0 0) ( 0 0 0)
2:Re ( 0 1 0) ( 0 0 0) ( 1 0 0) ( 0 0 0)
1m ( 0 0 0) ( 0 -1 0) ( 0 0 0) ( 0 0 0)
3:Re ( 0 0 1) ( 0 0 0) ( 0 0 -1) ( 0 0 1)
1m ( 0 0 0) ( 0 0 0) ( 0 0 0) ( 0 0 0)
4:Re ( -1 0 0) ( 0 0 0) ( 0 0 0) ( 0 0 0)
1m ( 0 0 0) ( -1 0 0) ( 0 1 0) ( 0 -1 0)
5:Re ( 0 1 0) ( 0 0 0) ( 0 0 0) ( 0 0 0)
1m ( 0 0 0) ( 0 1 0) ( 1 0 0) ( 0 -1 0)
6:Re ( 0 0 1) ( 0 0 0) ( 0 0 0) ( 0 0 0)
1m ( 0 0 0) ( 0 0 -1) ( 0 0 -1) ( 0 0 -1)
```

----- LINEAR COMBINATIONS of Basis Functions: coefficients u,v,w,p,q,....  
General expressions of the Fourier coefficients Sk(i) i=1,2,...nat

SYMM x,y,z Atom: Cu\_1 0.0000 0.0000 0.5000  
Sk(1): (u-p,v+q,w+z)

SYMM -x+1/2,-y,z-1/2 Atom: Cu\_2 0.5000 0.0000 0.0000  
Sk(2): i. (-u-p,-v+q,w-z)

SYMM y+3/4,-x+1/4,-z+3/4 Atom: Cu\_3 0.7500 0.2500 0.2500  
Sk(3): (v,-u,-w)+i. (q,p,-z)

SYMM -y+1/4,x+1/4,-z+3/4 Atom: Cu\_4 0.2500 0.2500 0.2500  
Sk(4): (v,-u,w)+i. (-q,-p,-z)

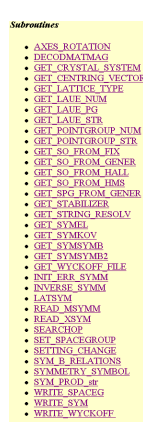
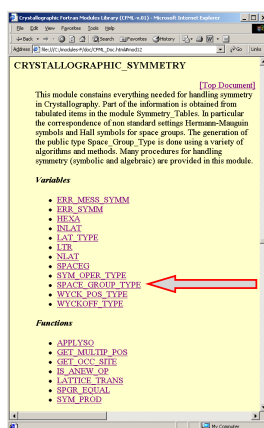
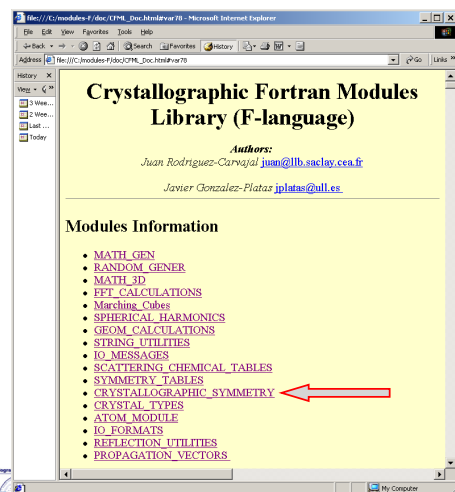


## Programming with CrysFML using it nearly as a black-box

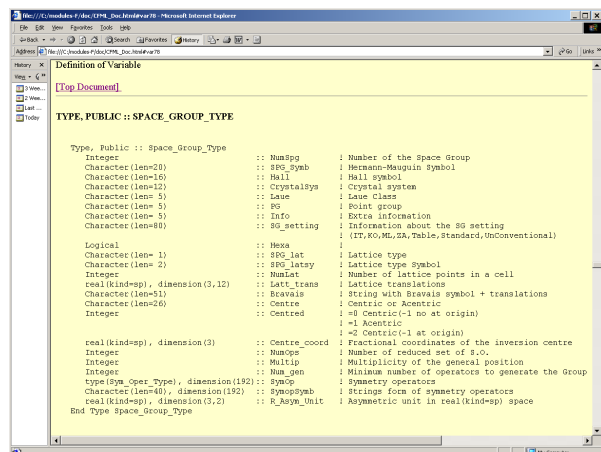
A Fortran 95/2003 compiler is needed (G95 is free!)

Learn the main structure types and procedures existing  
in the modules of the library by reading the  
documentation

Write a main program, using the modules of the library,  
for a particular purpose



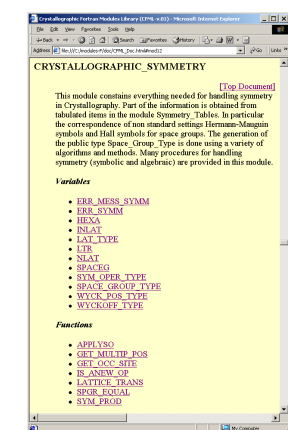
Example using  
the HTML  
automatically  
generated  
documentation



## Space Group Type in CrysFML

```
Type :: Space_Group_Type
Integer :: NumSpg ! Number of the Space Group
Character(len=20) :: HPG_Symb ! Hermann-Mauguin Symbol
Character(len=16) :: Hall ! Hall symbol
Character(len=12) :: CrystalSys ! Crystal system
Character(len=5) :: Laue ! Laue class
Character(len=5) :: PG ! Point group
Character(len=5) :: Info ! Extra information
Character(len=80) :: SG_setting ! Information about the SG setting
! (IT,KO,ML,2A,Table,Standard,UnConventional)
Logical :: Hexa ! Hexa
Character(len=1) :: SPG_lat ! Lattice type
Character(len=2) :: SPG_latasy ! Lattice type Symbol
Integer :: NumLat ! Number of lattice points in a cell
Integer :: Latt_trans ! Lattice translations
real(kind=sp), dimension(3,12) :: Bravais ! String with Bravais symbol + translations
Character(len=51) :: Centre ! Centric or Acentric
Character(len=26) :: Centred ! =0 Centric(-1 no at origin)
! =1 Acentric
! =2 Centric(-1 at origin)
real(kind=sp), dimension(3) :: Centre_coord ! Fractional coordinates of the inversion centre
Integer :: Number ! Number of reduced set of S.O.
Integer :: Multip ! Multiplicity of the general position
Integer :: Num_gen ! Minimum numb. of oper. to generate the group
type(Sym_Oper_Type), dimension(192) :: Symop ! Symmetry operators
Character(len=40) :: SymopSymb ! Strings form of symmetry operators
type(Wyckoff_Type), dimension(192) :: Wyckoff ! Wyckoff Information
real(kind=sp), dimension(3,2) :: R_Asym_Unit ! Asymmetric unit in real space
End Type Space_Group_Type
```

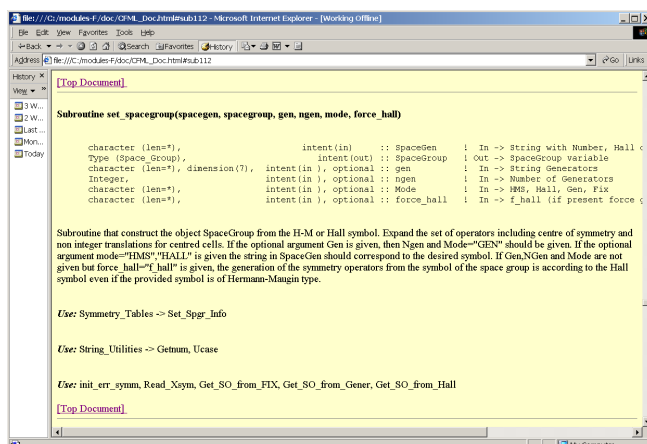




**Subroutines**

- AXES\_ROTATION
- DECOMMATMAG
- GET\_CRYSTAL\_SYSTEM
- GET\_CENTRING\_VECTORS
- GET\_LATTICE\_TYPE
- GET\_LAUE\_NDM
- GET\_LAUE\_PG
- GET\_LAUE\_STR
- GET\_POINTGROUP\_NDM
- GET\_POINTGROUP\_STR
- GET\_SO\_FROM\_FIX
- GET\_SO\_FROM\_GENER
- GET\_SO\_FROM\_HALL
- GET\_SO\_FROM\_HMS
- GET\_SPG\_FROM\_GENER
- GET\_STABILIZER
- GET\_STRING\_RESOLV
- GET\_SYMMET
- GET\_SYMMOV
- GET\_SYMSYMB
- GET\_SYMSYMB2
- GET\_WYCKOFF\_FILE
- INT\_FIX\_SYMM
- INVERSE\_SYMM
- LATSYM
- READ\_MSYMM
- READ\_NSYM
- SEARCHOP
- SET\_SPACEGROUP
- SETTING\_CHANGE
- SYM\_B\_RELATIONS
- SYMMETRY\_SYMBOL
- SYM\_PROD\_str
- WRITE\_SPACEGROUP
- WRITE\_SYM
- WRITE\_WYCKOFF

Example using the HTML automatically generated documentation



## The procedure *Set\_SpaceGroup*

```
Subroutine Set_Spacegroup (Spacegen, Spacegroup, Gen, Ngen, Mode, Force_Hall)
!-----Arguments -----!
character (len=*),          intent(in)          :: SpaceGen
type (Space_Group_Type),    intent(out)         :: SpaceGroup
character (len=*), dimension(:), intent(in), optional :: gen
integer,                   intent(in), optional :: ngen
character (len=*),          intent(in), optional :: Mode
character (len=*),          intent(in), optional :: force_hall
```

Header of the subroutine **Set\_Spacegroup**. Only two arguments are needed in the most simple cases.

The string **Spacegen** may contain the **Hermann-Mauguin (H-M) symbol**, the **Hall symbol** or, simply, the number of the space group.

The object **Spacegroup** is provided by a call to the subroutine.

## The procedure *Set\_SpaceGroup*

One can make a call to the subroutine as follows:

! Declarations omitted

```
Ngen=3
Gen(1)="y,-x, z"
Gen(2)="-x,-y,-z"
Gen(3)="x+1/2, y+1/2, -z"
Call Set_Spacegroup (Spacegen, Spacegroup, Gen, Ngen, "GEN")
```

On output the object **Spacegroup** of type **Space\_Group\_type** is filled with all possible information obtained from the list of the given generators.

## The procedure *Write\_SpaceGroup*

```
!-----
! Example of simple program using CrysFML
!-----
```

```
Program Get_SPG_info
use crystallographic_symmetry, only: &
space_group_type, set_spacegroup, write_spacegroup
character(len=20) :: spg_symb
type(space_group_type) :: SPG

do
write(unit=*,fmt="(a)",advance="no") &
" => Please enter a space group (H-M/Hall/number): "
read(unit=*,fmt="(a)") spg_symb
if(len_trim(spg_symb) == 0) exit
call set_spacegroup(spg_symb,SPG)
call write_spacegroup (SPG,full=.true.)
end do
stop
End Program Get_SPG_info
```

## The procedure *Write\_SpaceGroup*

The argument **full** in procedure **Write\_SpaceGroup** means that all detailed information in asked to be output in the screen. One may change the instruction to write directly to an already opened file. For instance writing:

```
Call Write_SpaceGroup (SPG,iunit=3,full=.true.)
```

directs the output to the file connected with logical unit 3



## Output of the small program: Get\_SPG\_info

```

Information on Space Group:
-----
> Number of Space group: 15
> Hermann-Mauguin Symbol: C 2/C
> Hall Symbol: -C 2ac
> Table Setting Choice: b1
> Setting Type: 1 (Generated from Hermann-Mauguin symbol)
> Crystal System: Monoclinic
> Lattice Class: 2/m
> Point Group: 2/m
> Bravais Lattice: C
> Lattice Symbol: mC
> Reduced Number of S.G.: 2
> General multiplicity: 8
> Centrosymmetry: Centric (-1 at origin)
> Generators (ext. -10,1):
>   Orthogonal unit: 0.000 < x < 0.500
>   Orthogonal unit: 0.000 < y < 0.500
>   Orthogonal unit: 0.000 < z < 0.500
> Centring vectors: 1
> Latt (1): ( 1/2, 1/2, 0 )

> List of all Symmetry Operators and Symmetry Symbols
> SPMG 1): x,y,z Symbol: 1
> SPMG 2): x,y,-z+1/2 Symbol: 2 0,y,1/4
> SPMG 3): x,y,z+1/2 Symbol: -1 0,0,0
> SPMG 4): x,y,-z+1/2 Symbol: 4 0,0,2
> SPMG 5): x+1/2,y+1/2,z Symbol: 1 (1/2,1/2,0)
> SPMG 6): x+1/2,y+1/2,-z+1/2 Symbol: 2 (0,1/2,0) 1/4,y,1/4
> SPMG 7): x+1/2,y+1/2,z+1/2 Symbol: -1 1/4,1/4,0
> SPMG 8): x+1/2,y+1/2,-z+1/2 Symbol: n (1/2,0,1/2) x,1/4,z

> Special Wyckoff Positions for C 2/C
> Mult Site Representative Coordinates (centring translations excluded)
> 4 d 1/4,1/4,1/2 3/4,1/4,0
> 4 c 1/4,1/4,0 3/4,1/4,1/2
> 4 b 0,1/2,0 0,1/2,1/2
> 4 a 0,0,0 0,0,1/2

> Please enter a space group (H-M/Hall/number):
  
```

## Another small program: test\_subgroup

```

Program test_subgroups
use crystallographic_symmetry, only: get_T_Subgroups, space_group_type, &
set_spacegroup, write_spacegroup, Lattice_trans, similar_transf_SG
use math_3d, only: determ_a

character(len=20) :: spg_sym
type(space_group_type) :: SpG, SpGn
real, dimension(3,3) :: trans
real, dimension(3) :: orig
integer :: nsg, i, j, ng, l
real :: det

do
write(unit=*,fmt="(a)",advance="no") " -> Please enter a space group (H-M/Hall/number): "
read(unit=*,fmt="(a)") spg_sym
if(len_trim(spg_sym) == 0) exit
call set_spacegroup(spg_sym, SpG)
write(unit=*,fmt="(a)",advance="no") " -> Please enter a transformation matrix: "
read(unit=*,fmt="(trans(i,:),1-1,3)
det=determ_a(trans)
write(unit=*,fmt="(a)",advance="no") " -> Please enter the new origin: "
read(unit=*,fmt="(a)") orig

call similar_transf_SG(trans,orig,SpG,SpGn) !Construct the subgroup of SpG that is compatible
!with the transformation matrix and change of origin
!give above

!Determine all subgroups of the new space group
call get_T_Subgroups(SpGn,Subgroup,nsg)

write(unit=*,fmt="(f,/,s,/)") " -> LIST of Translationengleiche Subgroups: "
do i=1,nsg
write(unit=*,fmt="(a,12,30a)") " -> ", Subgroup(i)%SpG_Sym, SubGroup(i)%hall,&
index: ["j,7"] -> ["", trim(SubGroup(i)%SymSpG_Sym(1))/"", "-1-1,ng-1,4",&
trim(SubGroup(i)%SymSpG_Sym(2))/"", trim(SubGroup(i)%SymSpG_Sym(3))/"", trim(SubGroup(i)%SymSpG_Sym(4))/""]
end do
end do
stop
end Program test_subgroups
  
```

## Output of the small program: test\_subgroup

```

Information on Space Group:
-----
=> Number of Space group: 176
=> Hermann-Mauguin Symbol: P 63/M
=> Hall Symbol: -P 6c
=> Table Setting Choice:
=> Setting Type: a'=a, b'=b, c'=c -> Origin: (0,0,0)
.....

=> LIST of Translationengleiche Subgroups:

=> P 63 P 6c Index: [ 2] -> { x,y,z : -}, Acentric
=> P 63/M -P 6c Index: [ 1] -> { x,y,z : -}, Centric
=> P 3 P 3 Index: [ 4] -> { x,y,z : -}, Acentric
=> P -3 P -3 Index: [ 2] -> { x,y,z : -}, Centric
=> P 1 1 21 P 2c Index: [ 6] -> { x,y,z : -}, Acentric
=> P 1 1 21/M -P 2c Index: [ 3] -> { x,y,z : -}, Centric
=> P -1 -P 1 Index: [ 6] -> { x,y,z : -}, Centric
=> unknown P -6c Index: [ 2] -> { x,y,z : -}, Acentric
=> unknown P -2c Index: [ 6] -> { x,y,z : -}, Acentric

=> Please enter a space group (H-M/Hall/number):
  
```

## Another Example: Check\_Group

```

Program Check_Group
use crystallographic_symmetry, only: Space_Group_Type, set_spacegroup
use reflections_utilities, only: hkl_Absent
use Symmetry_Tables, only: spgr_info, Set_Spgr_Info

..... ! Read reflections, apply criterion of "goodness" for checking,
! set indices i1,i2 for search in space group tables ...
..... ! omitted for simplicity

call Set_Spgr_Info()
do_group: do i=1,i2
hms=adjustl(spgr_info(i)%hkl)
hall=spgr_info(i)%hall
if(hms(i1)/="P" .and. .not. check_cent) cycle do_group ! Skip centred groups
call set_spacegroup(hall,Spacegroup,Force_Hall="y")
do j=1,nhkl
if(good(j) == 0) cycle !Skip reflections that are not good (overlap) for checking
if(absent(hkl(i,:),j), Spacegroup)
if(absent .and. intensity(j) > threshold) cycle do_group !Group not allowed
end do
! Passing here means that all reflections are allowed in the group -> Possible group!
num_group=num_group+1
write(unit=*,fmt=") " -> LIST OF POSSIBLE SPACE GROUPS, a total of "num_group" groups are possible"
write(unit=*,fmt=") " -----
write(unit=*,fmt=") " Number(IT) Hermann-Mauguin Symbol Hall Symbol"
write(unit=*,fmt=") " -----
do i=1,num_group(i)
j=num_group(i)
hms=adjustl(spgr_info(j)%hkl)
hall=spgr_info(j)%hall
num=spgr_info(j)%h
write(unit=*,fmt="(i10,4a)") numg," " ,hms," " ,hall
end do
end do
  
```

## Check\_Group output (1)

```

PROGRAM CHECK_GROUP: attempt to select the possible space groups from
an experimental Powder Diffraction Pattern

Author: J.Rodriguez-Carvajal (version 0.01, based on CrysFML)
  
```

```

Conditions:
Input hkl-file      : testgal.hkl
Crystal System      : Tetragonal
Check centred cells?: Y
Maximum angle       : 20.0000
Number of FWHMs     : 2.0000
Threshold in %      : 0.1000
  
```

=> List of read reflections:

h	k	l	Intensity	Sigma	2theta	FWHM	Good?
1	0	1	0.0000	0.0000	3.2518	0.0093	1
1	1	0	3.4230	0.4030	3.6146	0.0113	1
0	0	2	0.5280	0.2050	4.0212	0.0091	1
1	1	1	1.8570	0.3130	4.1363	0.0111	1
2	2	2	3562.2319	38.4840	8.2781	0.0138	1
2	1	3	5.4550	0.3910	8.3152	0.0118	0
3	1	1	23.2680	0.1620	8.3347	0.0124	0
0	0	4	0.0000	0.0000	8.4448	0.0102	0

## Check\_Group output (2)

```

=> Number of good reflections : 94
Maximum intensity : 3562.2319
Minimum (for observed) : 3.5622
Number of Space Group tested: 85
  
```

=> LIST OF POSSIBLE SPACE GROUPS, a total of 24 groups are possible

Number (IT)	Hermann-Mauguin Symbol	Hall Symbol
75	P 4	P 4
76	P 41	P 4w
77	P 42	P 4c
78	P 43	P 4cw
81	P -4	P -4
83	P 4/M	P -4
84	P 42/M	P -4c
89	P 4 2 2	P 4 2
90	P 4 21 2	P 4ab 2ab
91	P 41 2 2	P 4w 2c
92	P 41 21 2	P 4abw 2nw
113	P -4 21 M	P -4 2ab
114	P -4 21 C	P -4 2n
115	P -4 M 2	P -4 -2

```

Program Calc_structure_factors
use crystallographic_symmetry, only: space_group_type, Write SpaceGroup
use Atom Module, only: Atoms_List_Type, Write Atoms List
use crystal_types, only: Crystal_Cell_Type, Write Crystal_Cell
use Reflections_Uilities, only: Reflection_Type, Hkl_Uni, get_maxnumref
use IO_Formats, only: Readn_set_Xtal_Structure, err_mess_form, err_form
use Structure_Factor_Module, only: Structure_Factors, Write_Structure_Factors
type (space_group_type) :: SpG
type (Atoms_List_Type) :: A
type (Crystal_Cell_Type) :: Cell
type (Reflection_Type), allocatable, dimension(:) :: hkl
character(len=255) :: filcod 'Name of the input file'
real :: stlmax 'Maximum Sin(Theta)/Lambda'
integer :: MaxNumRef, Num, lun=1
do
  write(unit=*,fmt="(a)") " => Code of the file xx.cif (give xx): "
  read(unit=*,fmt="(a)") filcod
  if (len(trim(filcod)) == 0) exit
  write(unit=*,fmt="(a)") " => Maximum sinTheta/Lambda: "
  read(unit=*,fmt=*) stlmax
  open(unit=lun,file=trim(filcod)//".sfa", status="replace",action="write")
  call Readn_set_Xtal_Structure(trim(filcod)//".cif",Cell,SpG,A,Mode="CIF")

  if(err_form) then
    write(unit=*,fmt="(a)") trim(err_mess_form)
    exit
  else
    call Write_Crystal_Cell(Cell,lun)
    call Write_SpaceGroup(SpG,lun)
    call Write_Atoms_List(A,lun=lun)
    MaxNumRef = get_maxnumref(stlmax,Cell%CellVol,mult=SpG%Multip)
    if(allocated(hkl)) deallocate(hkl); allocate (hkl(MaxNumRef))

    call Hkl_Uni(Cell,SpG,.true.,0.0,stlmax,"s",Num,hkl)
    call Structure_Factors(A,SpG,Num,hkl,mode="NUC")
    call Write_Structure_Factors(lun,Num,hkl,mode="NUC")
  end if
end do
close(unit=lun)
End Program Calc_structure_factors

```

Program Calc\_structure\_factors

```

call Readn_set_Xtal_Structure(trim(filcod)//".cif",&
                             Cell,SpG,A,Mode="CIF")

```

Reads a CIF file and sets up the objects:

**Cell** : contains everything related to metrics

**SpG** : contains everything related to symmetry

**A** : contains everything concerned with atoms in the asymmetric unit

End Program Calc\_structure\_factors

Siemens 2009:  
University of



```

Program Calc_structure_factors
use crystallographic_symmetry, only: space_group_type, Write SpaceGroup
use Atom Module, only: Atoms_List_Type, Write Atoms List
use crystal_types, only: Crystal_Cell_Type, Write Crystal_Cell
use Reflections_Uilities, only: Reflection_Type, Hkl_Uni, get_maxnumref
use IO_Formats, only: Readn_set_Xtal_Structure, err_mess_form, err_form
use Structure_Factor_Module, only: Structure_Factors, Write_Structure_Factors
type (space_group_type) :: SpG
type (Atoms_List_Type) :: A
type (Crystal_Cell_Type) :: Cell
type (Reflection_Type), allocatable, dimension(:) :: hkl
character(len=255) :: filcod 'Name of the input file'
real :: stlmax 'Maximum Sin(Theta)/Lambda'
integer :: MaxNumRef, Num, lun=1
do
  write(unit=*,fmt="(a)") " => Code of the file xx.cif (give xx): "
  read(unit=*,fmt="(a)") filcod
  if (len(trim(filcod)) == 0) exit
  write(unit=*,fmt="(a)") " => Maximum sinTheta/Lambda: "
  read(unit=*,fmt=*) stlmax
  open(unit=lun,file=trim(filcod)//".sfa", status="replace",action="write")
  call Readn_set_Xtal_Structure(trim(filcod)//".cif",Cell,SpG,A,Mode="CIF")

  if(err_form) then
    write(unit=*,fmt="(a)") trim(err_mess_form)
    exit
  else
    call Write_Crystal_Cell(Cell,lun)
    call Write_SpaceGroup(SpG,lun)
    call Write_Atoms_List(A,lun=lun)
    MaxNumRef = get_maxnumref(stlmax,Cell%CellVol,mult=SpG%Multip)
    if(allocated(hkl)) deallocate(hkl); allocate (hkl(MaxNumRef))

    call Hkl_Uni(Cell,SpG,.true.,0.0,stlmax,"s",Num,hkl)
    call Structure_Factors(A,SpG,Num,hkl,mode="NUC")
    call Write_Structure_Factors(lun,Num,hkl,mode="NUC")
  end if
end do
close(unit=lun)
End Program Calc_structure_factors

```

```

Program Calc_structure_factors
use crystallographic_symmetry, only: space_group_type, Write SpaceG
use Atom Module, only: Atoms_List_Type, Write Atoms List
use crystal_types, only: Crystal_Cell_Type, Write Crystal_Cell
use Reflections_Uilities, only: Reflection_Type, Hkl_Uni, get_maxnumref
use IO_Formats, only: Readn_set_Xtal_Structure, err_mess_form, err_form
use Structure_Factor_Module, only: Structure_Factors, Write_Structure_Factors
type (space_group_type) :: SpG
type (Atoms_List_Type) :: A
type (Crystal_Cell_Type) :: Cell
type (Reflection_Type), allocatable, dimension(:) :: hkl
character(len=255) :: filcod 'Name of the input file'
real :: stlmax 'Maximum Sin(Theta)/Lambda'
integer :: MaxNumRef, Num, lun=1
do
  write(unit=*,fmt="(a)") " => Code of the file xx.cif (give xx): "
  read(unit=*,fmt="(a)") filcod
  if (len(trim(filcod)) == 0) exit
  write(unit=*,fmt="(a)") " => Maximum sinTheta/Lambda: "
  read(unit=*,fmt=*) stlmax
  open(unit=lun,file=trim(filcod)//".sfa", status="replace",action="write")
  call Readn_set_Xtal_Structure(trim(filcod)//".cif",Cell,SpG,A,Mode="CIF")

  if(err_form) then
    write(unit=*,fmt="(a)") trim(err_mess_form)
    exit
  else
    call Write_Crystal_Cell(Cell,lun)
    call Write_SpaceGroup(SpG,lun)
    call Write_Atoms_List(A,lun=lun)
    MaxNumRef = get_maxnumref(stlmax,Cell%CellVol,mult=SpG%Multip)
    if(allocated(hkl)) deallocate(hkl); allocate (hkl(MaxNumRef))

    call Hkl_Uni(Cell,SpG,.true.,0.0,stlmax,"s",Num,hkl)
    call Structure_Factors(A,SpG,Num,hkl,mode="NUC")
    call Write_Structure_Factors(lun,Num,hkl,mode="NUC")
  end if
end do
close(unit=lun)
End Program Calc_structure_factors

```

Program Calc\_structure\_factors

```

call Hkl_Uni(Cell,SpG,.true.,0.0,stlmax,&
             "s",Num,hkl)

call Structure_Factors(A,SpG,Num,hkl,mode="NUC")

call Write_Structure_Factors(lun,Num,hkl,&
                             mode="NUC")

```

**Hkl\_Uni** : Generates unique reflections in a  $\sin\theta/\lambda$  range (constructs, partially, the array of hkl objects)

**Structure\_Factors** : Completes the construction of the array of hkl objects

**Write\_Structure\_Factors** : Writes the results in a file

End Program Calc\_structure\_factors

Siemens 2009:  
University of



```

Program Calc_structure_factors
use crystallographic_symmetry, only: space_group_type, Write SpaceG
use Atom Module, only: Atoms_List_Type, Write Atoms List
use crystal_types, only: Crystal_Cell_Type, Write Crystal_Cell
use Reflections_Uilities, only: Reflection_Type, Hkl_Uni, get_maxnumref
use IO_Formats, only: Readn_set_Xtal_Structure, err_mess_form, err_form
use Structure_Factor_Module, only: Structure_Factors, Write_Structure_Factors
type (space_group_type) :: SpG
type (Atoms_List_Type) :: A
type (Crystal_Cell_Type) :: Cell
type (Reflection_Type), allocatable, dimension(:) :: hkl
character(len=255) :: filcod 'Name of the input file'
real :: stlmax 'Maximum Sin(Theta)/Lambda'
integer :: MaxNumRef, Num, lun=1
do
  write(unit=*,fmt="(a)") " => Code of the file xx.cif (give xx): "
  read(unit=*,fmt="(a)") filcod
  if (len(trim(filcod)) == 0) exit
  write(unit=*,fmt="(a)") " => Maximum sinTheta/Lambda: "
  read(unit=*,fmt=*) stlmax
  open(unit=lun,file=trim(filcod)//".sfa", status="replace",action="write")
  call Readn_set_Xtal_Structure(trim(filcod)//".cif",Cell,SpG,A,Mode="CIF")

  if(err_form) then
    write(unit=*,fmt="(a)") trim(err_mess_form)
    exit
  else
    call Write_Crystal_Cell(Cell,lun)
    call Write_SpaceGroup(SpG,lun)
    call Write_Atoms_List(A,lun=lun)
    MaxNumRef = get_maxnumref(stlmax,Cell%CellVol,mult=SpG%Multip)
    if(allocated(hkl)) deallocate(hkl); allocate (hkl(MaxNumRef))

    call Hkl_Uni(Cell,SpG,.true.,0.0,stlmax,"s",Num,hkl)
    call Structure_Factors(A,SpG,Num,hkl,mode="NUC")
    call Write_Structure_Factors(lun,Num,hkl,mode="NUC")
  end if
end do
close(unit=lun)
End Program Calc_structure_factors

```

## Installing and compiling CrysFML using G95 in Windows

### G95 in Windows using MinGW

Copy the file g95-MinGW.exe in a temporary directory and double-click on it: select the installation folder and say “yes” to all questions! (e.g. c:\G95, ... **warning! do not use “Program Files”**)

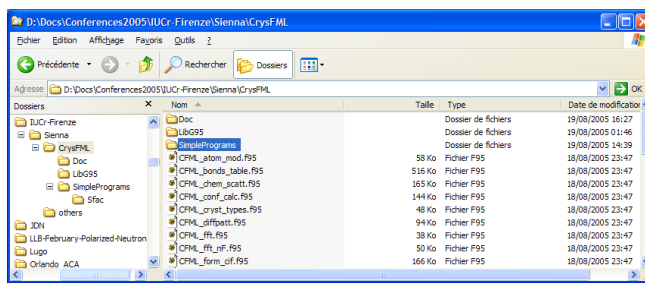
- Create a directory called “CrysFML” (e.g. c:\CrysFML)
- Copy the file CrysFML\_G95.zip in and extract all files respecting the directory structure
- Compile and build the library running the file “crysml\_g95.bat”

Sienna 2005: Crystallographic Computing School



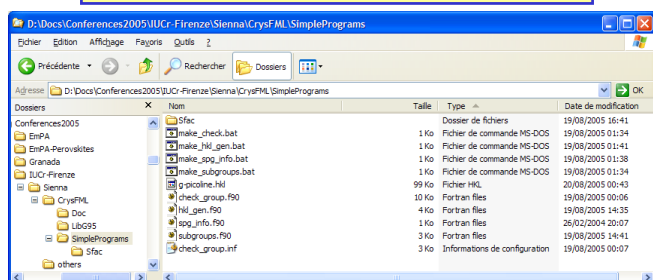
## The content of the CrysFML folder and sub-folders

All CrysFML files start with the prefix “CFML\_” and have extension .f95



## Content of the “SimplePrograms” folder

Four main programs and make\*.bat files, one \*.hkl file coming from FullProf, a \*.inf file and a sub-folder called “Sfac”



## Content of the “Sfac” folder

Source code files:

There are **two** modules:

“observed\_reflections” in file “**observ.f90**”

And

“cost\_functions” in file “**cost\_functions.f90**”

Three main programs:

“Calc\_structure\_factors” in “**sfac\_test.f90**”

“Optimizing\_structure” in “**Optim\_Sfac.f90**”

“Optimizing\_structure” in “**Opt\_restraints.f90**”

Sienna 2005: Crystallographic Computing School



## Input files for CrysFML (CIF and CFL)

```

Title NiFePO5
!
a      b      c      alpha  beta  gamma
Cell  7.1882  6.3924  7.4847  90.000  90.000  90.000
!
Space Group
!
Spggr  P n m a
!
!
x      y      z      B      occ  Spin Charge
Atom  Ni  Ni  0.0000  0.0000  0.0000  0.74  0.5  2.0  2.0
Atom  Fe  Fe  0.1443  0.2500  0.7074  0.63  0.5  5.0  3.0
Atom  P   P   0.3718  0.2500  0.1424  0.79  0.5  0.0  5.0
Atom  O1  O   0.3988  0.2500  0.64585 0.71  0.5  0.0  -2.0
Atom  O2  O   0.19415 0.2500  0.0253  0.70  0.5  0.0  -2.0
Atom  O3  O   0.0437  0.2500  0.4728  0.83  0.5  0.0  -2.0
Atom  O4  O   0.3678  0.0566  0.2633  0.77  1.0  0.0  -2.0
!
Codes for refinement
Vary xyz 0 1 0 1
!
Fix x_Fe y_O4
!
Equal y_Fe y_P 1.00
HKL-OBS mFe hkl
MIN-DSPACING 1.5
OPTIMIZE Fobs-Fcalc 1.0
SIM_ANN
!
Name of the cost function
CostNam FobsFcalc
!
T_ini anneal num_temps
!
TemParM 8.0 0.95 90
!
Nalgor Nconf nm_cycl num_therm accept
!
Algor_T 0 1 90 0 0.01
!
Value of Seed (if SeedVAL = 0, random seed)
SeedVAL 0
!
Treatment of initial configuration
InitCON RAN
    
```

Sienna 2005: Crystallographic Computing School



## Workshop: Connecting to hardware

### Goal of the exercises

The goal of this exercise is to try and abstract the different tasks to perform and separate them into different modules in the program. The exercises both simulate progress in the hardware and how it should be addressed, and in the demands that are made by the scientist using the application.

### General instructions

Try to solve these problems in such a way that a small change in the "hardware" to be controlled or a small change in the kind of "experiment" you want to do with the hardware will only need small changes in your code.

### Description of the environment

The "hardware" is represented in this exercise as a server that is running on port 7777 of a computer attached to the network. You can connect to it using a TCP socket connection. You send it an ASCII string command, followed by a carriage-return (ASCII 13). The server will give you an answer structured like:

- 1 character protocol-id ("1")
- 5 characters representing an integer status:
  - status = 0 = ERROR
  - status = 1 = OK
- 5 characters representing a block length (n)
- a carriage-return character
- n characters of data or n characters of error message

After the data block, the connection is closed by the server.

The server has an unknown number (call it x) of strings that it can return; these are numbered from 0 to x-1, and can be retrieved by sending a string representation of the number ("%d", followed by ASCII 13 as was said earlier) as a command to the server. Some other, secret, commands are available in the server as well, you will find out about these during the exercises.

## Socket programming

To complete this exercise the program must make an internet socket. The use of such a socket consists of a few steps:

- Create a socket. Parameters are:
  - "domain" is PF\_INET or AF\_INET
  - "type" is SOCK\_STREAM
  - "protocol" (if needed in your language) is 0
- Create a socket address from the host name or IP address and the port number
- Connect the socket to the address
- Use send() and recv() to communicate over the socket. IO

In case you have no experience programming with sockets, an example of how the first exercise can be solved is given on subsequent pages of this document. Examples are available in python and in C, and should not be used as an example of how to solve the exercise well, but only on how to use socket connections.

## The exercises

Exercise 1: Make a client that connects to the server, and retrieves and prints out string 0 (zero). That string is the next exercise.

## Example program in C

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <stdlib.h>

int port = 7777;

int s;
struct sockaddr_in serv_addr;
struct hostent *servent;

/* Receive a message from a socket based on knowledge of
the size of
    the packet to be received */
int recvfixed(int s, int n, char *x) {
    int i = 0;
    int tmp;
    while (i < n) {
        tmp = recv(s, x, n-i, MSG_WAITALL);
        if (tmp == -1) { return (-1); }
        i += tmp;
    }
    x[i] = '\0';
    return (0);
}

/* Open our socket and connect it to the server */
int sopen() {
    int s;
    s = socket(PF_INET, SOCK_STREAM, 0);
    servent = gethostbyname("127.0.0.1");
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = servent->h_addrtype;
    memcpy((char *) &serv_addr.sin_addr.s_addr, servent-
>h_addr_list[0],
        servent->h_length);
    serv_addr.sin_port = htons(port);
    connect(s, (struct sockaddr *)&serv_addr, sizeof
(serv_addr));
    return (s);
}

int main() {
    /* All the strings we need */
```

```

char protocol[2];
char s_status[6];
char s_length[6];
char s_sep[2];
char *mess;
/* Two of the strings correspond to integers */
int status, length;
/* Open the socket */
s = sopen();
/* Send the command 0 */
send(s, "0\r", 2, 0);
/* Interpret the protocolled answer */
if (recvfixed(s, 1, protocol)) {
    fprintf(stderr, "Could not receive protocol\n");
    return (1);
}
if (strncmp(protocol, "1", 1)) {
    fprintf(stderr, "Protocol mismatch\n");
    return (1);
}
if (recvfixed(s, 5, s_status)) {
    fprintf(stderr, "Could not receive status\n");
    return (1);
}
status = strtol(s_status, NULL, 10);
if (recvfixed(s, 5, s_length)) {
    fprintf(stderr, "Could not receive length\n");
    return (1);
}
length = strtol(s_length, NULL, 10);
if (recvfixed(s, 1, s_sep)) {
    fprintf(stderr, "Could not receive separator\n");
    return (1);
}
mess = malloc(length+1);
if (recvfixed(s, length, mess)) {
    fprintf(stderr, "Could not receive message\n");
    return (1);
}
if (status == 0) {
    fprintf(stderr, "ERROR: %s\n", mess);
    return (2);
} else {
    fprintf(stdout, "%s\n", mess);
}
return (0);
}

```

## Example program in Python

```
import sys,socket

host = 'localhost'
port = 7777

class Error(Exception):
    pass

def recvfixed(s, n):
    ret = ''
    while n > 0:
        str = s.recv(n)
        ret += str
        n -= len(str)
    return ret

def sopen():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    return s

def main():
    s = sopen()
    s.send("0\r")
    try:
        if recvfixed(s, 1) != '1':
            print >>sys.stderr, "Protocol mismatch"
            sys.exit(1)
    except IOError:
        print >>sys.stderr, "Could not receive protocol"
        sys.exit(1)
    try:
        status = int(recvfixed(s, 5))
    except IOError:
        print >>sys.stderr, "Could not receive status"
        sys.exit(1)
    try:
        length = int(recvfixed(s, 5))
    except IOError:
        print >>sys.stderr, "Could not receive length"
        sys.exit(1)
    try:
        sep = recvfixed(s, 1)
    except IOError:
        print >>sys.stderr, "Could not receive separator"
        sys.exit(1)
    try:
```



```

        mess = recvfixed(s, length)
    except IOError:
        print >>sys.stderr, "Could not receive message"
        sys.exit(1)
    if status == 0:
        print >>sys.stderr, "ERROR: %s" % mess
        sys.exit(2)
    else:
        print mess

if __name__ == "__main__":
    main()

```

## Tcl/Tk demo

Brian Toby

## Running the shell

- Tcl shell = /usr/bin/tclsh (tclsh.exe)
  - Tcl/Tk shell = /usr/bin/wish (wish.exe)
    - As usually distributed requires many extra files
- Here: combined distribution as one file (starkit)
- Includes several “extra” packages

## Starting “Tcl/Tk”

- Windows: ncnrpack-win.exe
- Linux: ncnrpack-linux
- OS X: ncnrpack-osx
  - Requires X11
  - Aqua version of Tcl/Tk does not support graphics demo

Can be run as

ncnrpack-xxx <script> **runs commands  
in script**

ncnrpack-xxx ***opens console***

## Tcl stuff

The set command

- set var1 value
- set var2 \$var1
- set var3 var1

The set command

- puts “writes a string: \$var1”
- puts {w/o substitution: \$var1}

## If-then-else

Embedded commands

- set list [glob \*]

Math

- set var [expr {pow(2,3)\*10}]

```
if {test1} {  
    statement(s)  
} else {  
    statement(s)  
}  
  
if {var1 == "test"} {  
    puts "test done"  
    set var1 ""  
} else {  
    puts "not test"  
}
```

## looping

```
foreach var {1 2 3 a b c} {  
    puts $var  
}  
  
for {set I 1} {$I <= 4} {incr I} {  
    puts $I  
}
```

## Tk background

- Master window is named “.”
- Something inside master is a “child”
- Name of children of master will be  
    .<name>
  - <name> ==> start w/lower case letter
  - .b or .bToby not .BT

## Some Tk Commands: making widgets

- Command label

```
label <child-name> -text "label text"  
– Returns <child-name>
```
- Command button

```
button <child-name> -text "txt" \  
    -command "tcl command"  
– Returns <child-name>
```
- N.B. creates child but does not display it

## Displaying widgets

- For very quick demos, use pack:

```
label .l -text sample  
pack .l
```
- Better, use grid

```
grid <child> -column <num> -row <num>  
  
label .l -text sample  
grid .l -column 1 -row 1
```

## Demo1

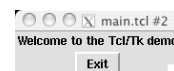
```
label .label -text "Welcome to the Tcl/Tk demo"  
grid .label -column 1 -row 1  
button .b -text "Exit" -command "exit"  
grid .b -column 1 -row 2
```

- Type the above into console *or*
- Windows: drag file demo1.txt onto **ncnrbpack-win.exe** icon
- *or* Command line:

```
Toby$ ncnrbpack-osx demo1.txt
```
- *or* In console (windows may need `cd <dir>` to get to right place):

```
source demo1.txt
```

## Demo 1



## Children with children

- Child windows  
`toplevel <name>`
- Frames (containers)  
`frame <name>`
- Child of child is named `.parent.child`  
`toplevel .win`  
`button .win.b -text "child button"`  
`pack .win.b`
- Destroy command deletes widgets & children  
`destroy .win` (deletes `.win` & `.win.b`)  
`destroy .` (same as `exit`)

## Demo 2

```
label .label -text "Welcome to the Tcl/Tk demo"
grid .label -column 1 -row 1
button .b -text "Exit" -command "exit"
grid .b -column 1 -row 2
toplevel .w
pack [label .w.l -text child]
pack [button .w.b -text kill -command "destroy .w"]
```

**Creates a 2nd window. Clicking on kill closes the child window; clicking on exit closes both**

## Demo 2



## Demo 3

```
foreach c {1 2 3} {
    foreach r {1 2 3} {
        grid [label .${c}${r} -text "${c}-${r}"] -column $c -row $r
    }
}
```

- Creates a table with grid

## Demo 3



## Demo 4

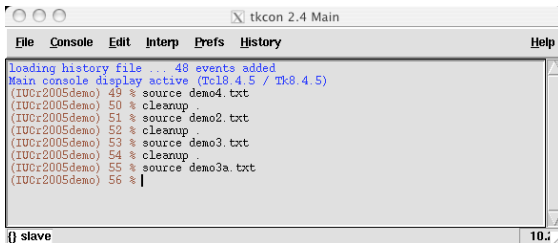
### Define a new command

```
# define a routine to delete children of an item
proc cleanup {parent} {
    foreach item [wininfo children $parent] {
        if {$item != ".tkcon"} {destroy $item}
    }
}
```

- Define a new command with:  
`proc <name> <args> {script}`  
– `<name> <arg(s)>` then executes command

## Demo 4

- Nothing happens!
  - New command has been defined: cleanup
  - Use as `cleanup . --` like this:

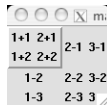


## Demo 3a

```
foreach c {1 2 3} {
  foreach r {1 2 3} {
    grid [label .${c}${r} -text "${c}-${r}"] -column $c -row $r
  }
}
# replace upper left element with frame
destroy .11
grid [frame .f -relief raised -bd 2] -column 1 -row 1
# fill the frame
foreach c {1 2} {
  foreach r {1 2} {
    grid [label .f.${c}${r} -text "${c}${r}"] -column $c -row $r
  }
}
```

- Creates a table inside a table with grid & frame

## Demo 3a



## Demo 3b

```
foreach c {1 2 3} {
  foreach r {1 2 3} {
    grid [label .${c}${r} -text "${c}-${r}"] -column $c -row $r
  }
}
# change an element on the fly to show it can be done
.33 configure -text "new element\n33"
```

- Changes an existing label

## Demo 3b

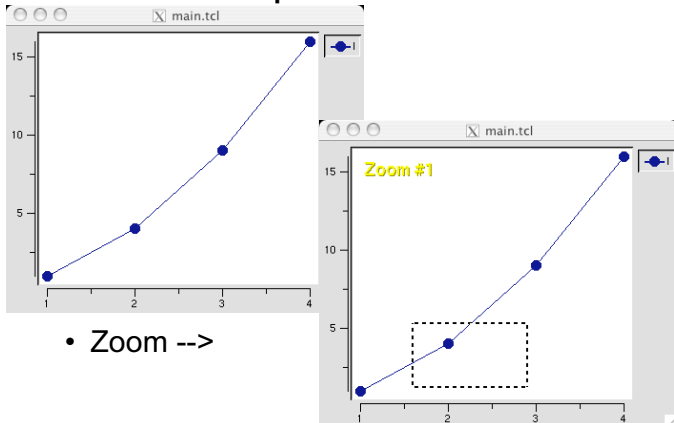


## Graphics demo

```
package require BLT
pack [blt::graph .g]
.g element create 1 -xdata {1 2 3 4} -ydata {1 2 9 16}
# oops
.g element configure 1 -ydata {1 4 9 16}
# allow zooming
Blit_ZoomStack .g
```

- Shows how to use BLT for plotting

## Graphics Demo



- Zoom -->

## Running external programs

- Many ways -- this one is platform independent:
 

```
set fp [open x.txt w]
puts $fp "input"
close $fp
exec program < x.txt > x.out
set fp [open x.out r]
while {[gets $fp line] >= 0} {
    #do something with line
}
```

## Running external programs

- Even better, have program write tcl code as output, then replace while with:
 

```
set lines [read $fp]
close $fp
eval $lines
```
- To suppress errors in output
 

```
catch {eval $lines}
```

## Running interactive programs: Win9x, -ME

```
# this creates a DOS box to run a program in
proc forknewterm {title command "wait 1" "scrollbar 1"} {
    global env expgui
    set pwd [file nativename [pwd]]

    # check the .EXP path -- can DOS use it?
    if {[string first // [pwd]] != -1} {
        MyMessageBox -parent . -title "Invalid Path" \
            -message {Error -- Use "Map network drive" to
                access this directory with a letter (e.g. F:) GSAS can't
                directly access a network drive} \
            -icon error -type ok -default ok \
            -helplink "expgui_Win_readme.html NetPath"
        return
    }
}
```

## Win-NT/-2000/-XP

```
if {[info command winutils::shell] == ""} {
    MyMessageBox -parent . -title "Setup error" \
        -message {Error -- "WINTILS not found. Can't do
            anything!"} \
        -icon error -type darn -default darn \
        -helplink "expgui_Win_readme.html Winexec"
    return
}
# loop over multiple commands
foreach cmd $command {
    # replace the forward slashes with backward
    regsub -all / $cmd \\ cmd
    winutils::shell [file join $dir gsastcl.bat] $cmd
}
}
```

```
proc forknewterm {title command} {
    global env expgui
    # loop over commands
    foreach cmd $command {
        # replace the forward slashes with backward
        regsub -all / $cmd \\ cmd
        exec $env(COMSPEC) /c \
            "start [file join $dir gsastcl.bat] $cmd"
    }
}
```

## Unix/OS X

```
proc forknewterm {title command "wait 1" "scrollbar 1"} {
    global env expgui
    set termopts {}
    if $scrollbar {
        append termopts " -sb"
    } else {
        append termopts " +sb"
    }
    if {$wait} {
        set suffix {}
    } else {
        set suffix {&}
    }
    catch {eval exec xterm $termopts -title [list $title] \
        -e /bin/sh -c [list $command] $suffix} errmsg
    if $expgui(debug) {puts "xterm result = $errmsg"}
}
```

- How can you tell where you are?

```
if {$tcl_platform(platform) == "windows" && \
    $tcl_platform(os) == "Windows 95"} {
    # win-95...
} elseif {$tcl_platform(platform) == "windows"} {
    # win-XP...
} else {
    # the rest
}
```

- See file expgui/gsascmds.tcl for more complete versions of the commands and for gsascmds.bat

# unit\_cell\_refinement.py

## Overview

The `unit_cell_refinement.py` [Python](#) example starts with a list of 2-theta peak positions derived from a powder diffraction experiment, and associated Miller indices obtained with an indexing program. The six unit cell parameters `a,b,c,alpha,beta,gamma` are refined to minimize the least-squares residual function:

```
sum over all Miller indices of (two_theta_obs -  
two_theta_calc)**2
```

The refinement starts with the unit cell parameters `a=10, b=10, c=10, alpha=90, beta=90, gamma=90`. A general purpose quasi-Newton [LBFGS](#) minimizer is used. The LBFGS minimizer requires:

- an array of *parameters*, in this case the unit cell parameters.
- the *functional* given the current parameters; in this case the functional is the residual function above.
- the *gradients* (first derivatives) of the functional with respect to the parameters at the point defined by the current parameters.

To keep the example simple, the gradients are computed with the finite difference method.

Before and after the minimization a table comparing `two_theta_obs` and `two_theta_calc` is shown. The script terminates after showing the refined unit cell parameters.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Input data

For simplicity, the input data are included near the beginning of the example script:

```
two_theta_and_index_list = """\n  8.81    0  1  1\n 12.23    0  0  2\n 12.71    0  2  0\n...\n 31.56    0  1  5\n 32.12    2  2  3\n""".splitlines()
```



Here two steps are combined into one statement. The first step is to define a multi-line Python string using triple quotes. In the second step the Python string is split into a Python list of smaller Python strings using the standard `splitlines()` method. It is instructive to try the following:

- Temporarily remove the `.splitlines()` call above and print `repr(two_theta_and_index_list)`. This will show a single string with embedded new-line characters:

```
•
• '      8.81    0  1  1\n      12.23    0  0  2\n      12.71    0  2  0\n ...
• 31.56    0  1  5\n      32.12    2  2  3\n'
```

- At the command prompt, enter [libtbx.help str](#) to see the full documentation for the Python `str` type including the documentation for `splitlines()`:

```
•
• | splitlines(...)
• |      S.splitlines([keepends]) -> list of
  | strings
• |
• |      Return a list of the lines in S,
  | breaking at line boundaries.
• |      Line breaks are not included in the
  | resulting list unless keepends
• |      is given and true.
```

- With the `.splitlines()` call added back, print `repr(two_theta_and_index_list)` again. This will show a Python list of strings:

```
•
• ['      8.81    0  1  1', '      12.23    0  0
  2', '      12.71    0  2  0', ...
• '      31.56    0  1  5', '      32.12    2  2  3']
```

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Conversion of input data to flex arrays

The list of Python strings as obtained above has to be converted to *flex arrays* to be suitable for calculating the residual sum. This is achieved with the following Python statements:

```
from cctbx.array_family import flex

two_thetas_obs = flex.double()
miller_indices = flex.miller_index()
for line in two_theta_and_index_list:
    fields = line.split()
    assert len(fields) == 4
    two_thetas_obs.append(float(fields[0]))
```

```
miller_indices.append([int(s) for s in fields[1:]])
```

The first statement imports the `cctbx.array_family.flex` module, which provides a number of array types, e.g. `int`, `double`, `std_string` and `miller_index`. These `flex` arrays can be one-dimensional or multi-dimensional. The numeric array types provide a comprehensive set of functions for element-wise operations such as `>`, `+`, `sin`, and reduction functions such as `sum`, `min`, `min_index`. Try [libtbx.help cctbx.array\\_family.flex](#) and [libtbx.help cctbx.array\\_family.flex.double](#) to see a listing of the available features. Almost all facilities provided by the `flex` module are implemented in C++ and are therefore very fast.

In the `unit_cell_refinement.py` example the input data are converted to the `flex` array types `double` and `miller_index`. The statement:

```
two_thetas_obs = flex.double()
miller_indices = flex.miller_index()
```

*instantiate* brand-new one-dimensional arrays. The initial size of both arrays is 0, as can be verified by inserting `print two_thetas_obs.size()` and `print miller_indices.size()`. The next three statements loop over all lines in `two_theta_and_index_list`:

```
for line in two_theta_and_index_list:
    fields = line.split()
    assert len(fields) == 4
```

Each line is split into fields. Use [libtbx.help str](#) again to obtain the documentation for the `split()` method. The `assert` statement reflects good coding practices. It is not strictly needed, but the following two statements assume that the input line consists of exactly four fields. If this is not the case, non-obvious error messages may result. Using `assert` statements is an easy way of obtaining more reasonable error messages. They also help others understanding the source code.

The `fields` are still strings, but it is easy to convert the first field to a Python `float` instance and to append it to the `two_thetas_obs` array:

```
two_thetas_obs.append(float(fields[0]))
```

The same result could be achieved in three steps:

```
s = fields[0] # Python lists are indexed starting with 0
v = float(s) # conversion from str -> float
two_thetas_obs.append(v)
```

However, the one-line version is not only shorter but clearly better for two main reasons:

- we don't have to invent names for the intermediate results (`s` and `v` in the second version); therefore the code is easier to read.
- the intermediate results are automatically deleted, i.e. the corresponding memory is released immediately.

A full equivalent of the one-line version is actually even longer:

```
s = fields[0]
v = float(s)
del s
two_thetas_obs.append(v)
del v
```

The conversion of the Miller indices is more involved. Three integer indices have to be converted from strings to integers and finally added to the `miller_indices` array. It could be done like this:

```
h = int(fields[1])
k = int(fields[2])
l = int(fields[3])
miller_indices.append([h,k,l])
```

However, anyone who has spent frustrating hours debugging silly copy-and-paste mistakes like `k = int(fields[1])` will prefer the alternative using Python's [List Comprehension](#) syntax:

```
hkl = [int(s) for s in fields[1:]]
```

This is equivalent to the longer alternative:

```
hkl = []
for s in fields[1:]:
    hkl.append(int(s))
```

Of course, that's hardly a gain over the simple copy-and-paste solution, but the list comprehension solution clearly is, and since it is *one* expression it can be directly used as an argument for `miller_indices.append()`.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Calculation of 2-theta angles

[Bragg's law](#) tells us how to compute diffraction angles `theta`. In typical textbook notation:

```
lambda = 2 * d_hkl * sin(theta)
```

In Python we cannot use `lambda` as a variable name since it is a reserved keyword for functional programming; therefore we use the variable name `wavelength` instead. Rearranging Bragg's equation leads to:

```
2 * sin(theta) = wavelength / d_hkl
```

I.e. we need to obtain two pieces of information, the `wavelength` and the *d-spacing* `d_hkl` corresponding to a Miller index `hkl`.

The input data in the `unit_cell_refinement.py` script are derived from a powder diffraction experiment with copper k-alpha radiation. A lookup table for the corresponding wavelength is compiled into the `cctbx.eltbx.wavelengths` module ([libtbx.help cctbx.eltbx.wavelengths.characteristic](#)):

```
from cctbx.eltbx import wavelengths
wavelength = wavelengths.characteristic("CU").as_angstrom()
```

We don't have to calculate `d_hkl` explicitly since the `cctbx.uctbx.unit_cell` object "knows" about Bragg's law and also how to compute `d_hkl` given unit cell parameters ([libtbx.help cctbx.uctbx.unit\\_cell](#)). This allows us to write:

```
from cctbx import uctbx
unit_cell = uctbx.unit_cell((10,10,10,90,90,90))
two_thetas_calc = unit_cell.two_theta(miller_indices,
wavelength, deg=True)
```

Conveniently the `two_theta()` method computes all `two_thetas_calc` in one call, given an array of `miller_indices`. Now that we have both `two_thetas_obs` and `two_thetas_calc` it is a matter of two lines to show a nice table:

```
for h,o,c in zip(miller_indices, two_thetas_obs,
two_thetas_calc):
    print "(%2d, %2d, %2d)" % h, "%6.2f - %6.2f = %6.2f" % (o,
c, o-c)
```

Use [libtbx.help zip](#) to learn about Python's standard `zip()` function, or consult the Python tutorial section on [Looping Techniques](#) for more information. The `print` statement can be understood by reading the tutorial section on [Fancier Output Formatting](#).

[[Complete example script](#)] [[Example output](#)] [[cctbx downloads](#)] [[cctbx front page](#)]  
[[Python tutorial](#)]

# Computation of the least-squares residual

Given `two_thetas_obs` and `two_thetas_calc`, the least-squares residual defined in the first section can be computed with three nested calls of functions provided by the `flex` module introduced before:

```
flex.sum(flex.pow2(two_thetas_obs - two_thetas_calc))
```

The inner-most call of a `flex` function may not be immediately recognizable as such, since it is implemented as an *overloaded* `-` operator. In a more traditional programming language the element-wise array subtraction may have been implemented like this:

```
element_wise_difference(two_thetas_obs, two_thetas_calc)
```

Python's operator overloading gives us the facilities to write `two_thetas_obs - two_thetas_calc` instead. This expression returns a new array with the differences. The `flex.pow2()` function returns another new array with the squares of the differences, and the `flex.sum()` function finally adds up the squared differences to return a single value, the least-squares residual. All intermediate arrays are automatically deleted during the evaluation of the nested expression as soon as they are no longer needed.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

# Gradient calculation via finite differences

The [Finite Difference Method](#) approximates the gradients of a function  $f(u)$  at the point  $x$  with the simple formula:

```
df/du = (f(x+eps) - f(x-eps)) / (2*eps)
```

`eps` is a small shift. This method is also applicable for computing partial derivatives of multivariate functions. E.g. the gradients of  $f(u, v)$  at the point  $x, y$  are approximated as:

```
df/du = (f(x+eps, y) - f(x-eps, y)) / (2*eps)
df/dv = (f(x, y+eps) - f(x, y-eps)) / (2*eps)
```

The main disadvantage of the finite difference method is that two full function evaluations are required for each parameter. In general it is best to use analytical gradients, but it is often a tedious and time consuming project to work out the analytical expressions. Fortunately, in our case we have just six parameters, and the runtime for one function evaluation is measured in micro seconds. Using finite differences is exactly the right approach in this situation, at least as an initial implementation.

To facilitate the gradient calculations, the residual calculation as introduced before is moved to a small function:

```
def residual(two_thetas_obs, miller_indices, wavelength,
            unit_cell):
    two_thetas_calc = unit_cell.two_theta(miller_indices,
                                          wavelength, deg=True)
    return flex.sum(flex.pow2(two_thetas_obs -
                              two_thetas_calc))
```

The finite difference code is now straightforward:

```
def gradients(two_thetas_obs, miller_indices, wavelength,
             unit_cell, eps=1.e-6):
    result = flex.double()
    for i in xrange(6):
        rs = []
        for signed_eps in [eps, -eps]:
            params_eps = list(unit_cell.parameters())
            params_eps[i] += signed_eps
            rs.append(
                residual(
                    two_thetas_obs, miller_indices, wavelength,
                    uctbx.unit_cell(params_eps)))
        result.append((rs[0]-rs[1])/(2*eps))
    return result
```

The `list()` constructor used in this function creates a copy of the list of unit cell parameters. The chosen `eps` is added to or subtracted from the parameter `i`, and a new `uctbx.unit_cell` object is instantiated with the modified parameters. With this the residual is computed as before. We take the difference of two residuals divided by `2*eps` and append it to the resulting array of gradients.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## LBFGS minimization

As outlined at the beginning, the `residual()` and `gradients()` are to be used in connection with the quasi-Newton [LBFGS](#) minimizer as implemented in the `scitbx.lbfgs` module. A minimal recipe for using this module is shown in the [scitbx/scitbx/examples/lbfgs\\_recipe.py](#) script:

```

class refinery:

    def __init__(self):
        self.x = flex.double([0])
        scitbx.lbfgs.run(target_evaluator=self)

    def compute_functional_and_gradients(self):
        f = 0
        g = flex.double([0])
        return f, g

    def callback_after_step(self, minimizer):
        pass

```

This `refinery` class acts as a `target_evaluator` object for the `scitbx.lbfgs.run()` function. Basically, the `target_evaluator` object can be any user-defined type, but it has to conform to certain requirements:

- `target_evaluator.x` must be a `flex.double` array with the parameters to be refined.
- `target_evaluator.compute_functional_and_gradients()` must be a function taking no arguments. It has to return a floating-point value for the functional, and a `flex.double` array with the gradients of the functional with respect to the parameters at the current point `target_evaluator.x`. The size of the gradient array must be identical to the size of the parameter array.
- `target_evaluator.callback_after_step()` is optional. If it is defined, it is called with one argument after each LBFGS step. The argument is an instance of the [scitbx::lbfgs::minimizer](#) C++ class and can be analyzed to monitor or report the progress of the minimization. `target_evaluator.callback_after_step` may or may not return a value. If the returned value is `True` the minimization is terminated. Otherwise the minimization continues until another termination condition is reached.

The `scitbx.lbfgs.run(target_evaluator=self)` call in the `__init__()` method above initiates the LBFGS procedure. This procedure modifies the `self.x` (i.e. `target_evaluator.x`) array in place, according to the LBFGS algorithm. Each time the algorithm requires an evaluation of the functional and the gradients, the `compute_functional_and_gradients()` method is called. I.e. the `refinery` object calls `scitbx.lbfgs.run()` which in turn calls a method of the `refinery` object. The term `callback` is often used to describe this situation. Note that both `compute_functional_and_gradients()` and `callback_after_step()` are callback functions; they are just called in different situations.

Based on the `residual()` and `gradients()` functions developed before, it is not difficult to customize the recipe for the refinement of unit cell parameters:

```

class refinery:

    def __init__(self, two_thetas_obs, miller_indices,
wavelength, unit_cell):
        self.two_thetas_obs = two_thetas_obs
        self.miller_indices = miller_indices
        self.wavelength = wavelength
        self.x = flex.double(unit_cell.parameters())
        scitbx.lbfgs.run(target_evaluator=self)

    def unit_cell(self):
        return uctbx.unit_cell(iter(self.x))

    def compute_functional_and_gradients(self):
        unit_cell = self.unit_cell()
        f = residual(
            self.two_thetas_obs, self.miller_indices,
self.wavelength, unit_cell)
        g = gradients(
            self.two_thetas_obs, self.miller_indices,
self.wavelength, unit_cell)
        print "functional: %12.6g" % f, "gradient norm: %12.6g" %
g.norm()
        return f, g

    def callback_after_step(self, minimizer):
        print "LBFGS step"

```

With this class all required building blocks are in place. The refinement is run and the results are reported with these statements:

```

refined = refinery(
    two_thetas_obs, miller_indices, wavelength,
    unit_cell_start)
print refined.unit_cell()

```

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## To try goes studying over

The title of this section is [Google's automatic translation](#) of the German saying "Probieren geht ueber studieren." It is an invitation to copy and run this and other example scripts. Insert `print` statements to develop a better understanding of how all the pieces interact. Use `print list(array)` to see the elements of `flex` arrays. It may also be useful to insert `help(obj)` to see the attributes and methods of `obj`, where `obj` can be any of the objects created in the script.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)



## Exercise (not very hard)

Read the 2-theta angles and Miller indices from a file.

Hint: Learn about `import sys` and `sys.argv`.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Exercise (not very hard)

`import iotbx.command_line.lattice_symmetry` and instantiate the `metric_subgroups` class with the refined unit cell.

Hint: Look for "P 1" in the `run()` function in `iotbx/iotbx/command_line/lattice_symmetry.py`.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Exercise (harder)

Estimate the start parameters from 6 indices and associated 2-theta angels.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Exercise (advanced)

Work out a way to use analytical gradients. The best solution is not clear to me (rwgk). You may have to refine metric tensor elements instead of the unit cell parameters to keep the problem manageable. See [d\\_star\\_sq\\_alternative.py](#) for a possible start. Compare the time it took you to work out the code with the analytical gradients to the time it takes to implement the finite difference method.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Exercise (requires creativity)

Study [adp\\_symmetry\\_constraints.py](#). Use:

```
constraints = sgtbx.tensor_rank_2_constraints(
```

```
space_group=space_group,  
reciprocal_space=False,  
initialize_gradient_handling=True)
```

to reduce the number of unit cell parameters to be refined.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

---

[View document source](#). Generated on: 2005-09-28 19:27 UTC. Generated by [Docutils](#) from [reStructuredText](#) source.

# direct\_methods\_light.py

## Overview

The `direct_methods_light.py` [Python](#) example is designed to read two [CIF](#) files from the [Acta Crystallographica Section C](#) web page as inputs:

- required: reduced X-ray diffraction data: [vj1132Isup2.hkl](#)
- optional: the corresponding refined structure: [vj1132sup1.cif](#)

The `iotbx.acta_c` module is used to convert the diffraction data to a `cctbx.miller.array` object; this is supported by James Hester's [PyCifRW](#) library. Normalized structure factors ("E-values") are computed, and the largest E-values are selected for phase recycling with the Tangent Formula.

The Miller indices of the largest E-values are used to construct index triplets  $h = k + h - k$  with the `cctbx.dmtbx.triplet_generator`. The Tangent Formula is repeatedly applied to recycle a phase set, starting from random phases. After a given number of cycles, the resulting phase set is combined with the E-values. The resulting Fourier coefficients are used in a Fast Fourier Transformation to obtain an "E-map". The E-map is normalized and a symmetry-aware peak search is carried out; i.e. the resulting peak list is unique under symmetry.

If the CIF file with the coordinates is given, it is first used to compute structure factors `f_calc`. The correlation with the diffraction data is shown. Next, the CIF coordinates are compared with the E-map peak list using the Euclidean Model Matching procedure (Emma) implemented in the `cctbx`. The resulting output can be used to quickly judge if the structure was solved with the simple Tangent Formula recycling procedure.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Recommended reading

The [unit\\_cell\\_refinement.py](#) example introduces some important basis concepts.

## Processing of `vj1132Isup2.hkl`

The first step is to get hold of the file name with the reduced diffraction data. The file name has to be specified as the first command-line argument:

```
iotbx.python direct_methods_light.py vj1132Isup2.hkl
```

In the example script, the command line argument is extracted from the `sys.argv` list provided by Python's standard `sys` module ([libtbx.help sys](#)):

```
import sys
reflection_file_name = sys.argv[1]
```

The `reflection_file_name` is used in the call of the `cif_as_miller_array()` function provided by the [iotbx.acta\\_c](#) module:

```
from iotbx import acta_c
miller_array =
acta_c.cif_as_miller_array(file_name=reflection_file_name)
miller_array.show_comprehensive_summary()
```

The `iotbx.acta_c` module makes use of the [PyCifRW](#) library to read CIF files. `PyCifRW` returns the CIF data items as plain strings. The `cif_as_miller_array()` function extracts the appropriate strings from the object tree returned by `PyCifRW` to construct an instance of the `cctbx.miller.array` class, which is one of the [central types in the cctbx source tree](#). The `miller.array` class has a very large number of methods ([libtbx.help cctbx.miller.array](#)), e.g. the `show_comprehensive_summary()` method used above to obtain this output:

```
Miller array info: vj1132Isup2.hkl:F_meas,F_sigma
Observation type: xray.amplitude
Type of data: double, size=422
Type of sigmas: double, size=422
Number of Miller indices: 422
Anomalous flag: False
Unit cell: (12.0263, 6.0321, 5.8293, 90, 90, 90)
Space group: P n a 21 (No. 33)
Systematic absences: 0
Centric reflections: 83
Resolution range: 6.01315 0.83382
Completeness in resolution range: 1
Completeness with d_max=infinity: 1
```

We can see that the `miller_array` contains data and sigmas, both of type double. It also contains Miller indices, an anomalous flag, a unit cell and a `space_group` object. These are the primary data members. The observation type is an optional annotation which is typically added by the creator of the object, in this case the `cif_as_miller_array()` function. The information in the last five lines of the output is calculated on the fly based on the primary information and discarded after the `show_comprehensive_summary()` call is completed.

Two other `cctbx.miller.array` methods are used in the following statements in the script:

```
if (miller_array.is_xray_intensity_array()):
    miller_array = miller_array.f_sq_as_f()
```

If the `miller_array` is an intensity array, it is converted to an amplitude array. The `f_sq_as_f()` method ("sq" is short for "square") returns a new `cctbx.miller.array` instance. At some point during the evaluation of the statement the old and the new instance are both present in memory. However, after the `miller_array = miller_array.f_sq_as_f()` assignment is completed, the old `miller_array` instance is deleted automatically by the Python interpreter since there is no longer a reference to it, and the corresponding memory is released immediately.

It is very important to understand that most `miller.array` methods do not modify the instance in place, but return new objects. The importance of minimizing the number of methods performing in-place manipulations cannot be overstated. In large systems, in-place manipulations quickly lead to unforeseen side-effects and eventually frustrating, time-consuming debugging sessions. It is much safer to create new objects. In most cases the dynamic memory allocation overhead associated with object creation and deletion is negligible compared to the runtime for the actual core algorithms. It is like putting on seat belts before a long trip with the car. The 10 seconds it takes to buckle up are nothing compared to the hours the seat belts protect you.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Computation of E-values

Having said all the things about the dangers of in-place operations, the next statement in the script happens to be just that:

```
miller_array.setup_binner(auto_binning=True)
```

However, the operation does not affect the primary data members of the `miller_array` (unit cell, space group, indices, data, sigmas). The `setup_binner()` call initializes or re-initializes a binner object to be used in subsequent calculations. The `binner` object is understood to be a secondary data member and its state only affects the results of future calculations. In situations like this in-place operations are perfectly reasonable.

The result of the `setup_binner()` call is shown with this statement:

```
miller_array.binner().show_summary()
```

The output is:

```
unused:      - 6.0133 [ 0/0 ]
bin  1: 6.0133 - 1.6574 [57/57]
bin  2: 1.6574 - 1.3201 [55/55]
bin  3: 1.3201 - 1.1546 [55/55]
```

```
bin 4: 1.1546 - 1.0496 [45/45]
bin 5: 1.0496 - 0.9747 [55/55]
bin 6: 0.9747 - 0.9175 [55/55]
bin 7: 0.9175 - 0.8717 [48/48]
bin 8: 0.8717 - 0.8338 [52/52]
unused: 0.8338 - [ 0/0 ]
```

This means we are ready to calculate quasi-normalized structure factors by computing `f_sq / <f_sq/epsilon>` in resolution bins:

```
all_e_values =
miller_array.quasi_normalize_structure_factors().sort(by_value="data")
```

This statement performs two steps at once. First, the `quasi_normalize_structure_factors()` method creates a new `cctbx.miller.array` instance with the same unit cell, space group, anomalous flag and Miller indices as the input `miller_array`, but with a new data array containing the normalized structure factors. The `sort()` method is used immediately on this intermediate instance to sort the E-values by magnitude. By default, the data are sorted in descending order (largest first, smallest last). This is exactly what we want here. To convince yourself it is correct, insert `all_e_values.show_array()`.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Generation of triplets

In direct methods procedures it is typical to generate the  $h = k + h-k$  Miller index triplets only for the largest E-values. In the example script, the largest E-values are selected with this statement:

```
large_e_values = all_e_values.select(all_e_values.data() >
1.2)
```

Again, this statement combines several steps into one expression. First, we obtain access to the array of all E-values via `all_e_values.data()`. This array is a `flex.double` instance, which in turn has its own methods ([libtbx.help cctbx.array\\_family.flex.double](#)). One of the `flex.double` methods is the overloaded `>` operator; in the `libtbx.help` output look for `__gt__()`. This operator returns a `flex.bool` instance, an array with `bool` values, `True` if the corresponding E-value is greater than 1.2 and `False` otherwise. The `flex.bool` instance becomes the argument to the `select()` method of `cctbx.miller.array`, which finally returns the result of the whole statement. `large_e_values` is a new `cctbx.miller.array` instance with the same unit cell, space group and anomalous flag as `all_e_values`, but fewer indices and corresponding data. Of the 422 E-values only 111 are selected, as is shown by this `print` statement:

```
print "number of large_e_values:", large_e_values.size()
```

At this point all the information required to generate the triplets is available:

```
from cctbx import dmtbx
triplets = dmtbx.triplet_generator(large_e_values)
```

The `triplet_generator` is based on the [cctbx::dmtbx::triplet\\_generator](#) C++ class which uses a very fast algorithm to find the Miller index triplets (see the references near the top of [triplet\\_generator.h](#)). The `triplets` object manages all internal arrays automatically. It is not necessary to know very much about this object, but is informative to print out the results of some of its methods, e.g.:

```
from cctbx.array_family import flex
print "triplets per reflection: min,max,mean: %d, %d, %.2f" %
(
    flex.min(triplets.n_relations()),
    flex.max(triplets.n_relations()),
    flex.mean(triplets.n_relations().as_double())
)
print "total number of triplets:",
flex.sum(triplets.n_relations())
```

Here the general purpose `flex.min()`, `flex.max()`, `flex.mean()` and `flex.sum()` functions are used to obtain summary statistics of the number of triplet phase relations per Miller index. `triplets.n_relations()` returns a `flex.size_t()` array with unsigned integers corresponding to the ANSI C/C++ `size_t` type. However, the `flex.mean()` function is only defined for `flex.double` arrays. Therefore `n_relations()` has to be converted via `as_double()` before computing the mean.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Tangent Formula phase recycling

Starting with [RANTAN](#), the predominant method for initiating Tangent Formula phase recycling is to generate random phases. In principle this is very easy. E.g. the `flex.random_double()` function could be used:

```
random_phases_rad =
flex.random_double(size=large_e_values.size())-0.5
random_phases_rad *= 2*math.pi
```

However, centric reflections need special attention since the phase angles are restricted to two values, `phi` and `phi+180`, where `phi` depends on the space group and the Miller index. A proper treatment of the phase restrictions is implemented in the `random_phases_compatible_with_phase_restrictions()` method of `cctbx.miller.array`:

```
input_phases =
large_e_values.random_phases_compatible_with_phase_restrictio
ns()
```

The underlying random number generator is seeded with the system time, therefore the `input_phases` will be different each time the example script is run.

The Tangent Formula recycling loop has this simple design:

```
result = input
for i in xrange(10):
    result = function(result)
```

In the example script the actual corresponding code is:

```
tangent_formula_phases = input_phases.data()
for i in xrange(10):
    tangent_formula_phases = triplets.apply_tangent_formula(
        amplitudes=large_e_values.data(),
        phases_rad=tangent_formula_phases,
        selection_fixed=None,
        use_fixed_only=False,
        reuse_results=True)
```

In this case `function()` is the `apply_tangent_formula()` method of the `triplet` object returned by the `cctbx.dmtbx.triplet_generator()` call. The function call looks more complicated than the simplified version because it requires a number of additional arguments customizing the recycling protocol. It may be interesting to try different settings as an exercise. See [cctbx::dmtbx::triplet\\_generator](#) for details.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## E-map calculation

Another `cctbx.miller.array` method is used to combine the `large_e_values` with the `tangent_formula_phases` obtained through the recycling procedure:

```
e_map_coeff = large_e_values.phase_transfer(
    phase_source=tangent_formula_phases)
```

The `phase_transfer()` returns a `flex.complex_double` array of Fourier coefficients. A general proper treatment of phase restrictions is automatically included, although in this case it just corrects for rounding errors.

Given the Fourier coefficients, an E-map could be obtained simply via `e_map_coeff.fft_map()`. However, we have to think ahead a little to address a



technical detail. A subsequent step will be a peak search in the E-map. For this we will use a peak search algorithm implemented in the `cctbx.maptbx` module, which imposes certain space-group specific restrictions on the gridding of the map. For all symmetry operations of the given space group, each grid point must be mapped exactly onto another grid point. E.g. in space group P222 the gridding must be a multiple of 2 in all three dimensions. To inform the `fft_map()` method about these requirements we use:

```
from cctbx import maptbx
e_map =
e_map_coeff.fft_map(symmetry_flags=maptbx.use_space_group_symmetry)
```

The resulting `e_map` is normalized by first determining the mean and standard deviation ("sigma") of all values in the map, and then dividing by the standard deviation:

```
e_map.apply_sigma_scaling()
```

Since maps tend to be large and short-lived, this is implemented as an in-place operation to maximize runtime efficiency. The `statistics()` method of the `e_map` object is used to quickly print a small summary:

```
e_map.statistics().show_summary(prefix="e_map ")
```

This output is of the form:

```
e_map max 11.9224
e_map min -2.68763
e_map mean -2.06139e-17
e_map sigma 1
```

Due to differences in the seed for the random number generator, the `max` and `min` will be different each time the example script is run. However, the `mean` is always very close to 0 since the Fourier coefficient with index (0,0,0) is zero, and `sigma` is always very close to 1 due to the prior use of `apply_sigma_scaling()`; small deviations are the accumulated result of floating-point rounding errors.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Peak search

Given the normalized `e_map`, the peak search is initiated with this statement:

```
peak_search =
e_map.peak_search(parameters=maptbx.peak_search_parameters(
    min_distance_sym_equiv=1.2))
```

The only purpose of the `maptbx.peak_search_parameters` class is to group the fairly large number of parameters ([libtbx.help](#) [cctbx.maptbx.peak\\_search\\_parameters](#)). This approach greatly simplifies the argument list of functions and methods involving peak search parameters. It also accelerates experimentation during the algorithm development process. Parameters can be added, deleted or renamed without having to modify all the functions and methods connected to the peak search.

In the example, the minimum distance between symmetry-related sites is set to 1.2 Å. This instructs the peak search algorithm to perform a cluster analysis. The underlying distance calculations are performed for symmetry-related pairs and pairs of peaks unique under symmetry ("cross peaks"). If the `min_cross_distance` peak search parameter is not specified explicitly (as in the example), it is assumed to be equal to the `min_distance_sym_equiv` parameter.

The cluster analysis begins by adding the largest peak in the map as the first entry to the peak list. All peaks in a radius of 1.2 Å around this peak are eliminated. The largest of the remaining peaks is added to the peak list, and all peaks in a radius of 1.2 Å around this peak are eliminated etc., until all peaks in the map are considered or a predefined limit is reached. The example uses:

```
peaks = peak_search.all(max_clusters=10)
```

to obtain up to 10 peaks in this way. The peaks are printed in this `for` loop:

```
for site,height in zip(peaks.sites(), peaks.heights()):
    print " (%9.6f, %9.6f, %9.6f)" % site, "%10.3f" % height
```

See the [unit\\_cell\\_refinement.py](#) example for comments regarding the standard Python `zip()` function. The Python tutorial section on [Fancier Output Formatting](#) is useful to learn more about the `print` statement.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Processing of `vj1132sup1.cif`

Since it is not easy to quickly judge from the peak list if the structure was solved, the `vj1132sup1.cif` file is used for verification purposes. It is processed in very much the same way as the `vj1132Isup2.hkl` file before:

```
coordinate_file_name = sys.argv[2]
```

```
xray_structure = acta_c.cif_as_xray_structure(
    file_name=coordinate_file_name,
    data_block_name="I")
```

The `cif_as_xray_structure()` call requires the name of the CIF data block name in addition to the file name. This is because Acta C coordinate CIF files may contain multiple structures (and because the `iotbx.acta_c` module is not sophisticated enough to simply "do the right thing" if the CIF file contains only one structure). The result is an instance of another [central type in the cctbx source tree](#), `cctbx.xray.structure`. The `xray_structure` object is best understood by asking it for a summary:

```
xray_structure.show_summary()
```

The output is:

```
Number of scatterers: 13
At special positions: 0
Unit cell: (12.0263, 6.0321, 5.829, 90, 90, 90)
Space group: P n a 21 (No. 33)
```

We can also ask it for a list of scatterers:

```
xray_structure.show_scatterers()
```

The result is:

Label	Scattering	Multiplicity	Coordinates	Occupancy	Uiso
O1	O	4	( 0.5896 0.4862 0.6045)	1.00	0.0356
O2	O	4	( 0.6861 0.2009 0.7409)	1.00	0.0328
C2	C	4	( 0.6636 0.2231 0.3380)	1.00	0.0224
H2	H	4	( 0.7419 0.1804 0.3237)	1.00	0.0270
C1	C	4	( 0.6443 0.3133 0.5818)	1.00	0.0222
C3	C	4	( 0.5923 0.0216 0.2916)	1.00	0.0347
H3A	H	4	( 0.6060 -0.0308 0.1387)	1.00	0.0520
H3B	H	4	( 0.5153 0.0605 0.3069)	1.00	0.0520
H3C	H	4	( 0.6104 -0.0930 0.3997)	1.00	0.0520
N1	N	4	( 0.6395 0.3976 0.1659)	1.00	0.0258
H1A	H	4	( 0.6815 0.5160 0.1942)	1.00	0.0390
H1B	H	4	( 0.5680 0.4351 0.1741)	1.00	0.0390
H1C	H	4	( 0.6544 0.3463 0.0261)	1.00	0.0390

Instead of writing:

```
xray_structure.show_summary()
xray_structure.show_scatterers()
```

we can also write:

```
xray_structure.show_summary().show_scatterers()
```

This approach is called "chaining". The trick is in fact very simple:

```
class structure:

    def show_summary(self):
        print "something"
        return self

    def show_scatterers():
        print "more"
        return self
```

Simply returning `self` enables chaining.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Correlation of F-obs and F-calc

Given the amplitudes F-obs as `miller_array` and the refined coordinates as `xray_structure`, F-calc amplitudes are computed with this statement:

```
f_calc = abs(miller_array.structure_factors_from_scatterers(
    xray_structure=xray_structure,
    algorithm="direct").f_calc())
```

This expression can be broken down into three steps. The first step is:

```
miller_array.structure_factors_from_scatterers(
    xray_structure=xray_structure,
    algorithm="direct")
```

This step performs the structure factor calculation using a direct summation algorithm (as opposed to a FFT algorithm). The result is an object with information about the details of the calculation, e.g. timings, or memory requirements if the FFT algorithm is used. If the details are not needed, they can be discarded immediately by extracting only the item of interest. In this case we use the `f_calc()` method to obtain a `cctbx.miller.array` instance with the calculated structure factors, stored in a `flex.complex_double` array. The outermost `abs()` function calls the `__abs__()` method of `cctbx.miller.array` which returns another new `cctbx.miller.array` instance with the structure factor amplitudes, stored in a `flex.double` array.

The correlation of F-obs and F-calc is computed with this statement:

```
correlation = flex.linear_correlation(f_calc.data(),
miller_array.data())
```

`flex.linear_correlation` is a C++ class ([libtbx.help](#)  
[cctbx.array\\_family.flex.linear\\_correlation](#)) which offers details about the correlation calculation, similar in idea to the result of the `structure_factors_from_scatterers()` above. We could discard all the details again, but the correlation coefficient could be undefined, e.g. if all values are zero, or if all values in one of the two input arrays are equal. We ensure the correlation is well defined via:

```
assert correlation.is_well_defined()
```

It is good practice to insert `assert` statements anywhere a certain assumption is made. The `cctbx` sources contain a large number of `assert` statements. They prove to be invaluable in flagging errors during algorithm development. In most situations errors are flagged close to the source. Time-consuming debugging sessions to backtrack from the point of a crash to the source of the problem are mostly avoided. Once we are sure the correlation is well defined, we can print the coefficient with confidence:

```
print "correlation of f_obs and f_calc: %.4f" %
correlation.coefficient()
```

It is amazingly high (0.9943) for the `vj1132` case.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Euclidean Model Matching (Emma)

Our goal is to match each peak site with a site in the `vj1132sup1.cif` file. To make this section less abstract, we start with an example result:

```
Match summary:
Operator:
  rotation: {{-1, 0, 0}, {0, -1, 0}, {0, 0, -1}}
  translation: {0.5, 0, -0.136844}
rms coordinate differences: 0.85
Pairs: 8
  O1 peak01 0.710
  O2 peak09 1.000
  C2 peak03 0.896
  C1 peak02 0.662
  C3 peak04 0.954
  H3B peak07 0.619
  H3C peak08 0.979
  H1C peak06 0.900
Singles model 1: 5
  H2   H3A   N1   H1A   H1B
```

```
Singles model 2: 2
peak00    peak05
```

This means, `peak01` corresponds to `01` in the CIF file with a mismatch of 0.710 Å, `peak09` corresponds to `02` with a 1.000 Å mismatch, etc. The match is obtained after inverting the hand of the `peaks` (the rotation) and adding `{0.5, 0, -0.136844}` to the coordinates (the translation). Some sites in the CIF files have no matching peaks (e.g. `N1`) and some peaks have no matching site in the CIF file (e.g. `peak00`). The overall RMS (root-mean-square) of the mismatches is 0.85. I.e. this match is not very good, except as a bad example.

In general, the comparison of two coordinate sets via pair-wise association of sites is quite complex due to the underlying symmetry of the search space. In addition to the space group symmetry, allowed origin shifts and a change of hand have to be taken into consideration. This is described in detail by [Grosse-Kunstleve & Adams \(2003\)](#).

The `cctbx.euclidean_model_matching` module is available for computing the pairs of matching sites. The search algorithm operates on specifically designed `cctbx.euclidean_model_matching.model` objects. I.e. we have to convert the `xray_structure` instance and the `peaks` to `cctbx.euclidean_model_matching.model` objects. Converting the `xray_structure` object is easy because the conversion is pre-defined as the `as_emma_model()` method:

```
reference_model = xray_structure.as_emma_model()
```

Converting the `peaks` object is not pre-defined. We have to do it the hard way. We start with assertions, just to be sure:

```
assert
reference_model.unit_cell().is_similar_to(e_map.unit_cell())
assert reference_model.space_group() == e_map.space_group()
```

This gives us the confidence to write:

```
from cctbx import euclidean_model_matching as emma
peak_model =
emma.model(special_position_settings=reference_model)
```

`special_position_settings` is a third [central type in the cctbx source tree](#). It groups the unit cell, space group, and the `min_distance_sym_equiv` parameter which defines the tolerance for the determination of special positions. `emma.model` inherits from this type, therefore we can use the `reference_model` (which is an `emma.model` object) anywhere a `special_position_settings` object is required. This is more convenient than constructing a new `special_position_settings` objects from scratch.

At this stage the `peak_model` object does not contain any coordinates. We add them with this loop:

```
for i,site in enumerate(peaks.sites()):
    peak_model.add_position(emma.position(label="peak%02d" % i,
site=site))
```

The loop construct is a standard idiom ([libtbx.help enumerate](#), [Looping Techniques](#)). `label="peak%02d" % i` creates a label of the form `peak000`, `peak001`, etc. The label and the `site` are used to construct an `emma.position` object which is finally added to the `peak_model` via the `add_position()` method.

The `emma.model_matches()` function computes a sorted list of possible matches:

```
matches = emma.model_matches(
    model1=reference_model,
    model2=peak_model,
    tolerance=1.,
    models_are_diffraction_index_equivalent=True)
```

The `tolerance` determines the maximum distance for a pair of a site in `model1` and a site in `model2`. The `models_are_diffraction_index_equivalent` parameter is used in the determination of the symmetry of the search space and has to do with indexing ambiguities. It is always safe to use `models_are_diffraction_index_equivalent=False`, but the search may be slower. If it is certain that the models are derived from the same diffraction data `models_are_diffraction_index_equivalent=True` can be used to reduce the runtime. In this case we are sure because the correlation between F-obs and F-calc is almost perfect.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Exercise (not very hard)

Run the example script several times. Each time the results will be different due to different random seeds. You will observe that Emma is often misled by the hydrogens in the CIF file. To solve this problem, modify the script to remove the hydrogens from the reference model.

Hint: Find the implementation of `cctbx.xray.structure.as_emma_model()` (`cctbx/cctbx/xray/__init__.py`). Note that `scatterer` has a `scattering_type` attribute.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Exercise (harder)

Compute the correlation between `all_e_values` and a `cctbx.xray.structure` constructed with the top 6 peaks, using `"const"` as the `scattering_type`.

Hint: Study `iotbx/iotbx/acta_c.py` to see how the `xray_structure` is constructed from the CIF file. However, use `peak_structure = xray.structure(special_position_settings=xray_structure)`. Study `cctbx.xray.structure.__init__()` to see why this works.

[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

## Exercise (advanced)

Refactor the example script by splitting it up into functions and possibly classes. Compute random starting phases a given number of times and repeat the Tangent formula recycling for each. Avoid duplicate work. I.e. don't read the inputs multiple times, don't create the Emma reference model multiple times.

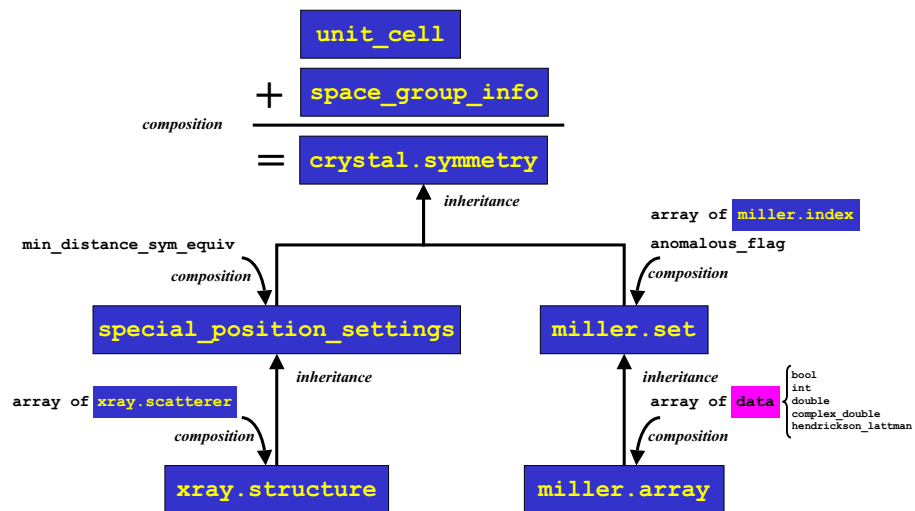
[\[Complete example script\]](#) [\[Example output\]](#) [\[cctbx downloads\]](#) [\[cctbx front page\]](#)  
[\[Python tutorial\]](#)

---

[View document source](#). Generated on: 2005-09-28 19:27 UTC. Generated by [Docutils](#) from [reStructuredText](#) source.



# Central cctbx types



## Tutorial on scoring methods for evaluation of electron density maps

[Tom Terwilliger](#), Los Alamos National Laboratory 12-August-2005

### What you will do in this tutorial

This tutorial will show you how you can combine scores from several criteria to evaluate the quality of an electron density map. You will be able to use real SAD data from a small protein (IF5a), compare the maps obtained with the correct, inverse, and random sites, and examine the scores obtained from analyzing features of the map such as its SD of local rms, its skew, or connectivity. You will be able to choose which scoring criteria are the most useful and to put these together into a simple scoring algorithm. You will then be able to apply your scoring algorithm to solve this structure and to test it on other structures.

### Getting ready

Download the developmental version of [PHENIX \(www.phenix-online.org\)](http://www.phenix-online.org) and install it. Installation takes just a minute or two if /usr/local/ is where you want phenix to be installed and you use the command:

```
./install --prefix=/usr/local/ --no-textal-maps
```

Set up your environment: put the line

```
source /usr/local/phenix-1.21a/phenix_env
```

into your .cshrc file in your home directory and use the C-shell: just type

```
csh
```

and the phenix environment should be set up. You can test it with

```
iotbx.reflection_statistics
```

which should tell you about the iotbx analysis tool which you may wish to use to look at the statistics of a data file.

### Copy over some data to work on

Make a new directory to work in and copy over the if5a data (called p9\_sad here) to it. For example:

```
cd
mkdir phenix_tutorial
cd phenix_tutorial
cp -r $PHENIX/examples/p9_sad .
cd p9_sad
pwd
ls
```

## Running AutoSol in PHENIX to phase and evaluate a map

Running AutoSol in PHENIX is easy. In this tutorial we will run it from a script. You control what AutoSol does with keywords in the file **Facts.list**. Edit the **Facts.list** that is in this directory to look like this:

```
force
create_scoring_table True
resolution 3.0
sg "I 4"
cell 113.949 113.949 32.474 90.000 90.000 90.00
read_sites True
ha_sites_file p9_sites.xyz
stop_after_scoring True
```

and also make a file **p9\_sites.xyz** with the correct heavy-atom sites for p9:

```
xyz      0.680      0.388      0.012
xyz      0.314      0.237      0.478
xyz      0.336      0.207      0.418
```

Now you can run AutoSol which will phase with these sites (and it will also phase with the inverse solution) and evaluate the maps from each:

```
phenix.runWizard AutoSol restart |tee AutoSol.log
```

You need the **restart** command so that it will start over from the beginning every time. As it runs notice that it gives you an output listing of the scores for randomized maps for each of several criteria, and then the scores for the maps calculated from your sites and from the inverse. Then it calculates a Z-score for your map for each criteria, where the Z-score is the score for your map minus the mean score for the random maps, divided by the standard deviation of scores for the random maps. The overall score for your map is the sum of these Z-scores.

Part of the output in **AutoSol.log** might look something like this:

```
Evaluate_solution
Setting up new scoring table with 6 values
TEMP7/resolve.scores CC 0.1449478
TEMP7/resolve.scores RFACTOR 0.6076099
TEMP7/resolve.scores SKEW -0.0040215286
```

where the three criteria CC, RFACTOR and SKEW are being used to evaluate the map.

You can also look at the file **AutoSol\_summary.dat** for a summary of the scoring on each of your maps (the one from the sites as you put them in and the inverse sites). Part of this file might look like:

```
Solution # 1    SCORE:58.8961885202 Dataset #1    FOM: 0.41 -----
-----
```

Solution 1 read from file p9\_sites.xyz Dataset #1

Score type:	CC	RFACTOR	SKEW	NCS_OVERLAP
Raw scores:	0.473	0.501	0.122	0.000
Z-scores:	36.014	18.782	4.100	0.000

Refined heavy atom sites (fractional):

xyz	0.680	0.388	0.012
xyz	0.314	0.237	0.478
xyz	0.336	0.207	0.418

The main goal in this tutorial is to come up with the best set of criteria for combining with Z-scores. You can use a single criterion or a group of them.

### Criteria for evaluating an electron density map

You have available to you several criteria for evaluating an electron density map. These are:

- SKEW -- the skew of the electron density map (related to  $\rho^{*3}$ )
- SD -- the standard deviation of local rms. This is the scoring criterion used in SOLVE. It is high if there are regions of low rms (solvent) and regions of high rms (protein), and low if the map has a uniform rms everywhere (random)
- RFACTOR -- the r-factor for 1 cycle of density modification. This reflects how well the observed amplitudes agree with those which would make the map conform to expectations (i.e., flat solvent).
- CC -- the correlation of the starting map and the map obtained with only map-probability phasing (phases from density modification not recombined with original phases). This criteria is closely related to RFACTOR.
- TRUNCATE -- The correlation of a map obtained by truncating the density in the map at a high level with the original map. A measure of how distinct the density is in the map.
- REGIONS -- The number of regions obtained if the map is truncated at a high level. A measure of the connectivity of the map.
- NCS\_OVERLAP -- the overlap of NCS-related density. This is zero if no NCS is present or cannot be found from the heavy-atom sites.

### Changing the criteria used for scoring

You can change the criteria used for scoring by adding a line to your **Facts.list** file specifying which criteria to use. Here is how you would choose SD and SKEW and RFACTOR:

```
score_type_list SD SKEW RFACTOR
```

Run AutoSol again, and see what scores you get for these criteria and how they add up. Now try all the scoring criteria. Decide how to put these together to get the best possible discrimination between the correct hand (the hand you put in) and the inverse.

Now reconsider what we are doing...is this the best way to optimize our scoring procedure? Perhaps we need to check it against a random set of sites? Perhaps the resolution matters? Perhaps we need also to check it against some different data?

Try a random set of sites by editing a new file **random.xyz** and putting some random coordinates in it. How does this fare in your scoring scheme?

Now try changing the resolution with the **resolution** command in your **Facts.list** file. The maximum resolution of this dataset is 2.1 Å.

### Trying your scoring criteria with a new set of data

Now try it with a new set of data. You can make a new directory for the sec17 data:

```
cd ../
mkdir sec17
cd sec17
cp $PHENIX/examples/sec17* .
pwd
ls
```

and a **Facts.list** something like this, except you will want to put in your own list of scoring criteria:

```
force
create_scoring_table True
resolution 3.0
read_sites True
ha_sites_file sec17.xyz
score_type_list TRUNCATE REGIONS CC RFACTOR SKEW SD
stop_after_scoring True
```

and a **sec17.xyz** like this:

```
xyz      0.3780      0.1961      -0.0047
xyz      0.1489      0.4676      0.0266
xyz      0.1411      0.4397      0.4504
```

Does your scoring scheme still work? Is it the best one for this dataset? Perhaps you might optimize your scoring scheme to simultaneously be good for both datasets.

### Use your scoring scheme to solve the IF5a structure

Now that you have an optimized scoring scheme, use it to solve the IF5a structure. Make a new **Facts.list** that looks like this one, but with your scoring criteria:

```
force
create_scoring_table True
build True
```

```
ha_iteration False
resolution 2.5
sg "I 4"
cell 113.949 113.949 32.474 90.000 90.000 90.00
score_type_list CC RFACTOR SKEW
```

If you want to have the AutoSol Wizard iterate in finding additional heavy-atom sites and optimizing their positions using difference Fourier's after density modification, try adding the lines

```
ha_iteration True
max_ha_iterations 1
fix_xyz_after_denmod False
```

to your **Facts.list** file.

Now run AutoSol again now using the **restart** command as before:

```
phenix.runWizard AutoSol restart |tee AutoSol.log
```

You should obtain an **AutoSol\_summary.dat** that lists the best solution found and the scores you obtained with it and where the model is located. You can look at the model with the command :

```
phenix.pymol
```

You can also look at all your results with a GUI if you type

```
phenix
```

and answer "yes" to the questions asked and then select **Wizards** and then double-click on **AutoSol** and then click on **RUN** in the lower right corner to reset the Wizard. You can then click on the **magnifying glass** icon at the top of the Wizard and it will give you some options of things to look at (phasing log, model, model and map, etc).

---

## Call for Contributions to the Next CompComm Newsletter

The seventh issue of the Compcomm Newsletter is expected to appear around April of 2006 with the tentative primary theme involving “Minimisation” algorithms for indexing, structure solution and refinement. 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, historical articles, programming languages, techniques and other news.

Please send articles and suggestions directly to the editor.

***Lachlan M. D. Cranswick***

Canadian Neutron Beam Centre

National Research Council of Canada

Building 459, Station 18,

Chalk River Laboratories,

Chalk River, Ontario,

Canada, K0J 1J0

E-mail: [lachlan.cranswick@nrc.gc.ca](mailto:lachlan.cranswick@nrc.gc.ca)

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