Commission on Crystallographic Computing and Commission on Crystallographic Teaching
International Union of Crystallography
http://www.iucr.org/resources/commissions/crystallographic-computing
http://www.iucr.org/resources/commissions/crystallographic-teaching
Computing Newsletter No. 10 / Teaching Newsletter No. 3, November 2009

This issue's theme:

Age Concern

http://www.iucr.org/resources/commissions/crystallographic-computing/newsletters
http://www.iucr.org/resources/commissions/crystallographic-teaching/newsletters

Table of Contents

(This issue's editor: Lachlan Cranswick)

(Warning – unless you want to kill 138 pages worth of forest – DO NOT press the "print" button. For hardcopies – you may like to only print out the articles of personal interest.)

Following historical teaching material as a separate "Addendum A" PDF file to this issue

Following manuals and reference materials as a separate "Addendum B" PDF file to this issue

Chair: Lachlan M. D. Cranswick
Canadian Neutron Beam Center (CNBC),
National Research Council of Canada (NRC),
Building 459, Station 18,
Chalk River Laboratories,
Chalk River, Ontario,
Canada, K0J 1J0
Tel: (613) 584-8811 (Ext. 43719)
Fax: (613) 584-4040
E-mail: lachlan.cranswick@nrc.gc.ca

Dr Benedetta Carrozzini
Institute of Crystallography (IC)
National Research Council (CNR)
Via G. Amendola, 122/o
70126 Bari - Italy
Tel. +39 080 5929 147
Fax +39 080 5929 170
E-mail: benedetta.carrozzini@ic.cnr.it
WWW: http://www.ic.cnr.it/carrozzini.php

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: rwgk@cci.lbl.gov
WWW: http://cctbx.sourceforge.net/
WWW: http://www.phenix-online.org/
WWW: http://cci.lbl.gov/~rwgk/

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

Prof Jordi Rius
Dpt. of Crystallography
Institut de Ciencia de Materials de Barcelona, CSIC
Campus de la UAB
08193-Bellaterra, Catalunya, Spain
Tel: +93 580 18 53
Fax: +93 580 57 29
E-mail: jordi.rius@icmab.es

Dr K. Sekar
Bioinformatics Centre
(Centre of Excellance in Structural Biology and Bio-computing)
Supercomputer Education and Research Centre
Indian Insitute of Science
Bangalore 560 012
India
Tel: 91-(0)80-23601409, 22933059, 22932469
Fax: 91-(0)80-23600683, 23600551
E-mail: sekar@physics.iisc.ernet.in and sekar@serc.iisc.ernet.in
WWW: http://www.physics.iisc.ernet.in/~dichome/sekhome/

Prof Peter Turner
Crystal Structure Analysis Facility,
School of Chemistry (F11),
University of Sydney,
Sydney, NSW,
Australia 2006.
E-mail: turner_p@chem.usyd.edu.au and p.turner@chem.usyd.edu.au
WWW: http://csaf.chem.usyd.edu.au/home_nographics.htm

Prof Björn Winkler
Inst. f. Geowissenschaften / FE Mineralogie / Abt. Kristallographie
Universitaet Frankfurt
Altenhoeferallee 1
D-60438 Frankfurt am Main, Germany
Phone: +49 69 798 40107 / 40108 (secretary)
FAX : +49 69 798 40109
E-mail: b.winkler@kristall.uni-frankfurt.de

Associate Prof. Dr. Min Yao
Laboratory of X-ray structural biology,
Faculty of Advanced Life Science,
Hokkaido University,
060-0810, Sapporo, Japan
Tel: +81-(0)11-706-4481
Fax: +81-(0)11-706-4481
E-mail: yao@castor.sci.hokudai.ac.jp
WWW: http://altair.sci.hokudai.ac.jp/g6/index-e.html

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
WWW: http://www.physics.mcmaster.ca/?page=sw://lists/Minibio_2004.php4?ID=4

Dr Graeme Day
Department of Chemistry,
University of Cambridge,
Lensfield Road, Cambridge, CB2 1EW,
United Kingdom
tel: +44 (0)1223-336390
fax: +44 (0)1223-336362
E-mail: gmd27@cam.ac.uk
WWW: http://www.ch.cam.ac.uk/staff/gmd.html

Computing Commission Consultants (cont'd)

Prof Santiago García-Granda
Faculty of Chemistry,
University Oviedo,
C/ Julian Claveria, 8
33006 Oviedo - Asturias, Spain
Tel. +34-985104061
Fax +34-985103125
Mobile: +34-690029092
E-mail: sgg@uniovi.es
WWW: http://www.uniovi.es/xray/

Prof Alessandro Gualtieri
Università di Modena e Reggio Emilia,
Dipartimento di Scienze della Terra,
Via S.Eufemia, 19,
41100 Modena, Italy
Tel: +39-059-2055810
Fax: +39-059-2055887
E-mail: alessandro.gualtieri@unimore.it
WWW: http://www.terra.unimo.it/en/personaledettaglio.php?user=alex

Prof Bart Hazes
Department of Medical Microbiology and Immunology
1-41 Medical Sciences Building
University of Alberta
Edmonton, AB T6G 2H7
Canada
E-mail: Bart.Hazes@Ualberta.ca
WWW: http://www.ualberta.ca/~mmi/faculty/bhazes/bhazes.html

Dr James Hester
Bragg Institute, ANSTO,
PMB 1, Menai, New South Wales, 2234,
Australia
Tel: +61 2 9717 9907
Fax: +61 2 9717 3145
E-mail: jxh@ansto.gov.au and jamesrhester@gmail.com
WWW:
http://www.ansto.gov.au/research/bragg_institute/contacts/dr_james_hester

Prof 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
WWW: http://www.protein.osaka-u.ac.jp/rcsfp/supracryst/

Dr Lukas Palatinus
Laboratory for Crystallography,
Swiss Federal Institute of Technology
BSP - Dorigny
CH-1015 Lausanne
Switzerland
Tel.: +41 (0)21 693 0639
Fax: +41 (0)21 / 6 93 05 04
E-mail: palat@fzu.cz

Dr Riccardo Spagna
Institute of Crystallography - CNR
Seat of Monterotondo
Area della Ricerca Roma 1
Via Salaria Km 29.3
00016 Monterotondo Stazione (Roma) Italy
Office tel.: +39.06 90672 614
Office fax : +39.06 90672 630
Cellular : +39 339 2766710
E-mail: riccardo.spagna@gmail.com and riccardo.spagna@ic.cnr.it
WWW: http://www.ic.cnr.it/ and http://www.ic.cnr.it/spagna.php

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

Chair: Prof. Paola Spadon
Department of Chemical Sciences
Padua University, Via Marzolo 1
35131 Padova, Italy
Tel: +39049 8275275
Fax: +39049 8275239
E-mail: paola.spadon@unipd.it

Roberto Baggio
Depto.de Fisica (Lab.TANDAR),
Comision Nacional de Energia Atomica,
Av. Gral Paz 1499, 1650 San Martín,
Pcia. de Buenos Aires,
Argentina
Tel: (54-11-)6772 7109
Fax: (54-11-)6772 7121
E-mail: baggio@cnea.gov.ar

Professor Elena V. Boldyreva
Head of the Chair of Solid State Chemistry,
Novosibirsk State University,
Pirogova, 2,
Novosibirsk 630090,
Russia
Tel: +7-3833-397238
Fax: +7-3833-397845
E-mail: boldyrev@nsu.ru

Professor Gervais Chapuis
Swiss Federal School of Technology,
BSP, CH-1015 Lausanne
Switzerland
Tel/Fax: +41(0)21 693 0630/0634
E-mail: Gervais.chapuis@epfl.ch

Lachlan M. D. Cranswick
Canadian Neutron Beam Center (CNBC),
National Research Council of Canada (NRC),
Building 459, Station 18, Chalk River Laboratories,
Chalk River, Ontario,
Canada, K0J 1J0
Tel: (613) 584-8811 (Ext. 43719)
Fax: (613) 584-4040
E-mail: Lachlan.Cranswick@nrc.gc.ca

Professor Katherine Kantardijeff
Chemistry and Biochemistry/Center for Molecular Structure,
California State University Fullerton,
800 N. State College Blvd.,
Fullerton, CA, 92834-6866, USA
Tel: 1(657)278-3752 ; Fax: 1(657)939-4225
E-mail: kkantardjieff@fullerton.edu

Dr Melanie Rademeyer
School of Chemistry, Univ. of Kwazulu-Natal
Private Bag X01
Scottsville 3209,
South Africa
E-mail: rademeyerm@ukzn.ac.za

Professor Miriam Rossi
Department of Chemistry
Vassar College,
Poughkeepsie, NY, 12604-0484,
USA
E-mail: rossi@vassar.edu

Dr Siegbert Schmid
School of Chemistry
The University of Sydney,
Sydney NSW 2006,
Australia
E-mail: s.schmid@chem.usyd.edu.au

Dr Hidehiro Uekusa
Department of Chemistry and Materials Science,
Tokyo Institute of Technology
Ookayama 2, Meguro-ku, Tokyo, 152-8551,
Japan
Tel & Fax +81-3-5734-3529
E-mail: uekusa@cms.titech.ac.jp

Consultants

Dr Enrique Espinosa
Laboratoire de Cristallographie et de Modélisation des Matériaux Minéraux et Biologiques
LCM3B (UMR UHP -CNRS 7036),
Faculté des Sciences et Techniques, BP 239,
Boulevard des Aiguillettes, 54506 Vandoeuvre-lès-Nancy,
France
Tel : +33-(0)383-684953, Fax: +33-(0)383-406492
E-mail: Enrique.Espinosa@lcm3b.uhp-nancy.fr

Professor T.N. Guru Row
Solid State and Structural Chemistry Unit
Indian Institute of Science
Bangalore, 560 012,
India
Tel: +91-80-22932796;+91-80-22932336 Fax: +91-80-23601310
E-mail: ssctng@sscu.iisc.ernet.in

Dr Marco Milanesio
DISTA, Univ. Piemonte Orientale
Via Bellini 25 G
15100 Alessandria,
Italy
Tel: 0039 0131 287414 Fax: 0039 0131 287416
E-mail: marco.milanesio@mfn.unipmn.it

Age Concern - the Background

Judith A. K. Howard[1] & David J. Watkin[2]
1) Chemistry Department, Durham University, Durham, DH1 3LE, United Kingdom; 2) Chemical Crystallography, Chemistry Research Laboratory, Oxford, OX1 3PD, United Kingdom. E-mail: david.watkin@chem.ox.ac.uk

Age Concern is a software project involving programmers in Durham and Oxford. As the project approaches maturity, it seems appropriate to explain some of the events leading to its formation.

DJW was Chairman of the IUCr Computing Commission from 1999 to 2002, having been a member of the committee for some time before, and served ex-officio afterwards. During his chairmanship the signs of the deteriorating condition of small-molecule crystallographic computing, which had started some years before, became clearer - difficulties in scheduling Computing micro-symposia, lack of any programming skills at all in the new generation of crystallographers, the slow march towards retirement of the major generation of crystallographic programmers.

In 2002, at the IUCr XIX congress held in Geneva, DJW gave a paper "Crystallographic Computing: Where Do We Go Now?" (Acta Cryst (2002), A58, (Supplement) C254). In this he observed that traditionally, crystallographic software has been developed and then maintained by the authors, usually at their institution's expense. He also pointed out, perhaps obviously, that many of the authors of significant programs are now approaching retirement and it was by no means clear who would replace them. In a world where many institutions are being forced to make harsh cost analyses there were serious issues to be faced. The following questions were posed:

1. Is there a problem emerging, or will 'cultural evolution' ensure survival of the best?
2. How can Crystallographic Software be financed and maintained?
3. What role can government institutions (EPSRC,CNR,CNRS,NSF etc) play in software projects?
4. Who should 'own' academic software, and what provisions can be made for its maintenance?
5. What can companies do with software which becomes unprofitable?
6. Is there a future for academic software?
7. Is there a risk of us loosing 'software diversity'?
8. Is software moving towards 'the best possible', or 'the lowest acceptable' level?

If it worth preserving legacy software, and if so, who should pay?

In the summing up it was remarked that while the role of computers in research was now inescapable, there was no good model to explain who was going to pay for the software we all need.

To try to get a communal discussion started, and to raise awareness of the growing problems, a Computing Commission Open Meeting "The Future of Crystallographic Software" was organised as part of the Congress, with introductory talks be George Sheldrick (author of several important programs), Phil Hastings from Accelrys (representing a commercial point of view), and Howard Flack (representing the IUCr). George, as ever optimistic, was sure that events would self-regulate and that there was no need for concern. Phil was concerned about many things, including funding the development of potentially interesting industrial applications. There was a massive cost in taking the science, theory, algorithms developed in an academic environment, and turning them into robust and maintainable commercial solutions. On the issue of maintaining legacy software, he asked:

- What should companies do with software which becomes unprofitable?
- Is it worth preserving legacy software, and if so, who should pay?

and replied:

- The concept of obsolescence in the software world is somewhat abstract
- There is need to understand why we want to preserve legacy code and what it is that needs to be preserved
- Commercial companies would in general prefer to retain some control over their own legacy code
- Cost of maintenance can exceed the cost of the initial development.

Howard explained that the IUCr was aware of the problems, but was not in a position to favour and sponsor any individual branch of crystallography. Its over-riding concern was the maintenance of standards in its journals.

The following year, 2003, Ross Angel organised a micro-symposium "SP.02: Future Strategies for Successful Crystallographic Computing" at the American Crystallographic Association Annual Meeting in Cincinnati. The report on the session is appended **(Appendix 1)**. DJW gave the opening talk - "Crystallographic Software - a Bleak Future?" but it was Jim Pflurath's contribution "There is No Such Thing as Free Software." which effectively summed up the session, and in effect pointed out that those of us who had worked during the 60's to 80's had lived in the halcyon days of software development, and that things would never be the same again.

During this time, Judith Howard (Durham) had also been worried by the same issues. The BCA /CCG Intensive teaching schools had been running successfully since 1987 and these involved both DJW and JAKH as the two originators of these courses, principally for chemical crystallographers. The idea evolved from discussions in 1985 in the Turin ECM, when we were already concerned that some of the basic skills were being eroded by the very excellence of the programs we were all beginning to use. The course material was concerned with practical fundamentals, and deliberately kept "software neutral" since we hoped to avoid the risk of losing broad computational and mathematical skills from future generations of crystallographers. These schools are still running successfully at Durham in alternate years and have been continuously remodelled to take account of changes in the technologies [e.g. CCDs] as well as changing software. We are very pleased to see that there has been a resurgence of interest in crystallographic computing in the younger generations in recent years.

DJW & JAKH with other tutors and teachers on the BCA schools, and colleagues generally at different meetings, continued to debate and increasingly questioned whether this was serious problem, whether our students were loosing ability and interest in computational aspects of the subject or were we just worrying unnecessarily?

Over time, we discussed this with our biological friends and they too began to see a problem in their community, with increasingly sophisticated black boxes driving their diffractometers, increasing numbers of data sets with more synchrotrons on line and cryo-freezing of protein samples. The reaction of the BSG was to hold courses tailored to bio-physical needs in the years alternate to the Durham course. Now

the BSG runs one each year and there are many similar academic courses run worldwide, to maintain the skills set of the next generations of structural scientists. Durham colleagues now also run a successful Powder diffraction school.

Collaboration.

At the 2004 BCA Spring meeting in York, DJW & JAKH felt that the time was right for an attempt to consolidate some aspects of small molecule crystallographic computing. We held a small informal meeting with both chemical and biological creators of crystallographic software at which we reached the consensus that there was a problem, but not so clearly how to address it. During the remainder of that year we collected information and ideas, and in 2004 **(Appendix 2)** Luc Bourhis (Durham), Bob Cernik (Daresbury Lab), Jeremy Cockcroft (Birkbeck College, London), Richard Cooper (Oxford), Kevin Cowtan (York), Lee Daniels (Rigaku, USA), Eleanor Dodson FRS (York), Oleg Dolomanov (Nottingham), John Evans (Durham), Louis Farrugia (Glasgow), Chris Gilmore (Glasgow), Jason Green (EPSRC), Rob Hooft (Delft), Judith Howard FRS (Durham), Simon Parson (Edinburgh), Horst Puschmann (Durham), George Sheldrick FRS (Gotttingen), Ton Spek (Utrecht), Janette Thomas (CELS), David Watkin (Oxford) held a round table discussion of the issues. Eleanor Dodson explained the situation in macromolecular crystallography, and how the EPSRC Collaborative Computing Project, CCP4, was evolving to embrace modern programming techniques, in particular benefiting from electronic communications and the internet to enable distributed contribution to a community based project. The small molecule software development community is much more fragmented, leading to problems with information and strategy exchange between systems. **(Appendix 3)**

Spurred on by the support of this meeting, the Oxford and Durham teams worked together on preparing the case for a major collaborative EPSRC grant proposal. The objectives were far-reaching but we believed and hoped that they were feasible. The proposal was submitted in October 2004, and an award was made to Durham and Oxford Universities in May the following year (2005). Thus the 'Age Concern ' project was born. Contributions from other members of the team can be read in this edition of the Newsletter. The objectives are given in **Appendix 4** and a full report will appear in 2011 after the project has been completed.

Age Concern - the Background : Appendix 1

ACA, Cincinnati, July 2003-08-01 : Micro symposium SP.02: Future Strategies for Successful Crystallographic Computing. Organised by Ross Angel & David Watkin

SP.02.01 Crystallographic Software – a Bleak Future? David Watkin & Richard Cooper

David Watkin, restricting his comments to the small molecule situation, tried to foresee developments in this field. He noted that most widely used programs come from authors in the last half of their expected careers, that past experience had shown software to fade away rapidly after their authors retirement, and that there appeared to be no new young authors in posts that would enable them to continue to maintain programs. Although George Sheldrick was more optimistic about the emergence of new programmers, Watkin was worried that much of the wealth of experience encoded in current programs would be lost to future programmers because little of it was documented. He pinned his hope for the future on small molecule developments being made to the open-sourced structured programming now emerging from the macromolecular sector.

SP.02.02 Government Funded Central Initiatives for Encouraging Diversity of Freely Available Crystallographic Software; and the threat of Crystallographic Software Patents. Lachlan Cranswick.

Lachlan Cranswick explained the role and functioning of the EPSRC funded CCP14 project for Single Crystal and Powder Diffraction. Before Lachlan's time, this project had sought to maintain established software on multiple platforms. Developments in the World Wide Web and the reduced choice of hardware platforms meant that this was no longer necessary, and under Lachlan's stewardship CCP14 had taken on a role aimed at raising peoples awareness of the available software and exposing the strong points of different programs. His mission had been to encourage people to seek out the best tool for each problem, rather than just make do with the software they were familiar with. His close interaction with users at the many workshops he has run has enabled him to provide valuable feedback to people still developing programs.

The second half of Lachlan's talk was distinctly worrying. He presented a whole catalogue of cases where ideas that were in common circulation had been successfully patented, mainly in the USA. The situation looks as if it will deteriorate further should Europe also introduce Software Patents.

SP.02.03 Hacker Vulnerability: A Major New Complication in Crystallographic Computing. Carroll Johnson.

Carroll Johnson briefly explained the methods used by hackers to gain access to private and institutional computers, and what can be done to minimise the risks of intrusion. He explained that the main risk was to machines left running on local area networks, especially homegrown networks. The principal safe guards are to use firewalls, and to run intrusion detection and logging software. He pointed out the need to verify that backup systems really would enable one to recover work after a serious hack or system crash.

SP.02.04 Crystallographic Software from the NIST Centre for Neutron Research. Brian Toby.

Brian Toby looked back over 20 years of programming, and lamented the fact that with each generation, fewer and fewer people are able to write even the simplest programs. He observed '*One thing has not changed: support for programming efforts remains lacklustre*'. Brian tried to provide answers to three questions: What are the software needs within the community? How should programming efforts be organised? and What are the incentives to make these goals happen?

SP.02.05 There is No Such Thing as Free Software. Jim Pflurath, Rigaku/MSC Inc.

Jim Pflugrath painted an excellent summary of the history of software development, the similarities and differences between writing code in laboratories and in a company, and the mechanisms by which software can be funded. At the end of the day, he pointed out, crystallographic software is generally paid for by the taxpayer, either through the salaries of university staff, through grants for individual projects or as part of the purchase of an instrument. The so-called free software is generally paid for by some ones employer (even if they don't realise it). In later discussion it emerged that software written at home in the evenings but related to what is done at work, also generally belongs to the employer!

Like Brian Toby, Jim also noted that programming skills are disappearing amongst scientists, to the point where many will not even be able to read code with any understanding. He drew parallels between writing software in a company and in a university. In particular, he noted that in both environments the programmers are also usually scientists, working in collaboration with other scientists. One common difference was the measures taken by companies to ensure some level of quality control and security.

Looking to the future, he explained that Rigaku/MSC make a point of employing scientists that can both write code, and teach other programmers the science of crystallography. '*After all*', he said, '*there is more to software than 'push a button"*

SP.02.06 Crystallographic Computing at Bruker Nonius. Susan Byram, Bruker Advanced X-ray Solutions.

Susan Byram showed examples of recent software products that were the result of collaborations outside of the company, and explained that interaction with the university community was vital. She took the opportunity to explain that while Bruker-Nonius did protect their investment in instrument design with patents, to date they had not patented software or algorithms. Bruker-Nonius are keenly aware that users or their funding agencies will not accept the real cost of software development. '*Frequent comments are heard that no funding is available to purchase software for more than a few hundred dollars, or to support software upgrades*'. Her response to the observation that less and less software is being developed in the public sector was to suggest that grant applications should contain an amount to cover the real cost of writing innovative software by commercial or independent developers.

SP.02.07. The CrysAlis Software Suite for Area and Point Detector Measurements: Open Source Option and User Modifications. Mathias Meyer, Oxford Diffraction, Poland Sp.

Oxford Diffraction is a relatively new player in the field, and so Mathias Meyer took the opportunity to provide an overview of their software products, and explain how they hope to encourage active interaction with the user community. The CrysAlis software is open source, so that users can examine it and make modifications. The modified code remains local to the user, but is dynamically linked to the rest of the system as a so-called plug-in. This mechanism enables enthusiastic users to try out and develop new ideas without undermining the security of the rest of the code.

SP.02.08 Developing Modern Crystallographic Libraries and Applications: PHENIX and the Computational Crystallographic Toolbox. Ralf Grosse-Kunstleve, Nigel Moriarty, Nicholas Sauter & Paul Adams.

Paul Adams explained how the need for rapid innovation in macromolecular software to meet the demands of the structural genomics efforts required a totally new approach to system development. The PHENIX project hopes to achieve this through a two-layered approach. High level applications are written in PYTHON to facilitate rapid and accurate code development, while the underlying core algorithms are written in C++, the computational crystallographic toolbox (cctbx). This toolbox is freely available under an Open Source license with the aim of fostering widespread collaborations.

Age Concern - the Background : Appendix 2

Durham-Program-04: The Future of Crystallographic Software : A workshop to be held on the 23rd of May 2004 in Trevelyan College, Durham

Aim:

The aim of this workshop is to focus the concerns about the future of crystallographic software harboured by all those wishing to attend into one or two clear statements. These statements will provide a sound basis for the successful application for the funding necessary to act in time before the foreseen problems manifest themselves with potentially catastrophic consequences.

Strategy:

This is a workshop, not a day of seminars. We hope that all those attending will have prepared a brief outline of the problem and how they see it, together with possible solution. Apart from some leading keynote speakers, who will focus on their special area of expertise, we hope to hear many more short presentations during the day, hopefully leading to many in-depth discussions.

Provisional outline of the Program for the Workshop:

Introduction/welcome [JAKH/HPILB]

Keynote Speakers

Dr. David Watkin, Oxford and Prof George Sheldrick, Goettingen:
Introductory talk

Prof Ton Spek, Utrecht
Software in the area of Small Molecule Crystallography

ProfBill David, RAL
Software in the area of Powder Diffraction

Prof Eleanor Dodson FRS, York
Software in the area of Protein Crystallography

Open discussion:

with specific questions to be raised and answered (hopefully), Some discussion of the technical points [but don't want to get bogged down here]

Lunch [buffet] may happen at any time .... and we just keep going .....

Further wide ranging discussions

Formulation of the Statements, funding options, grant applications, wider picture of wealth creation through jobs in the area and beyond the crystallographic community.
Re-creating the crystallographic programming spirit in the younger community/training, Identification/selection of obvious research/leadership/management roles so that we can proceed from here with real aims and goals and people taking responsibility for given task/areas.

End by 4-5pm so people can get home Sunday night.

Age Concern - the Background : Appendix 3

Durham-Report-04: The Future of Crystallographic Software : report of workshop held on the 23rd of May 2004 in Trevelyan College, Durham

Sponsorship for this workshop from CELS and GSK (Glaxo Smith Kline) is gratefully acknowledged.

A workshop on the topic in the title was held on the 23rd of May in Trevelyan College, Durham [Programme attached]

Participants attending :-

Luc Bourhis (Durham) Bob Cernik (Daresbury Lab) Jeremy Cockcroft (Birkbeck College, London), Richard Cooper (Oxford) Kevin Cowtan (York) Lee Daniels (Rigaku, USA) Eleanor Dodson FRS (York) Oleg Dolomanov (Nottingham),John Evans (Durham), Louis Farrugia (Glasgow), Chris Gilmore (Glasgow) Jason Green (EPSRC), Rob Hooft (Delft) Judith Howard FRS (Durham) Simon Parson (Edinburgh), Horst Puschmann (Durham) George Sheldrick, FRS (Gotttingen) Ton Spek (Utrecht), Janette Thomas (CELS) David Watkin (Oxford)

Participants who had hoped to be there but who in the event were unable to attend:

Bill David (RAL), Carmelo Giacovazzo (Bari), Vaclav Petricek (Prague) and also representatives from Oxford Diffraction, PanAnalytical, EPSRC and GSK

The workshop follows from a very small discussion group on the same theme at the York BCA Meeting (April 2003) and it was called by us [JAKH/HP/LB] - a small group of the Durham crystallography lab, who only very recently became involved in any software development - and yet almost everyone 'major' who has been involved in crystallographic programming for many years or decades, came to Durham for this one-day workshop. We believe that this demonstrates a considerable amount of interest and concern, no matter what people might declare as individuals.

George Sheldrick (Author of SHELX).

George's short talk was an historic perspective to begin with and an interesting summary of what he thinks the main concerns should be and which topics should be discussed in more detail. He raised the issues of open source codes and there was some concern expressed in general discussion about patenting of software, the changes being brought in by European legislation and the differences that do exist in this area between Europe and the US already. He also raised the question of the role being played and to be played by the instrument manufacturers in the overall picture.

There really wasn't anything very controversial in what he said and he provided a very good overview.

He tends to believe that there is no acute problem, no funding crisis per se, for students and post-docs, while he agrees and he is rather concerned that there is little career structure available for crystallographers who want to follow these future areas of research. He acknowledges that it cannot follow the career pattern of an academic such as himself and others in the audience, of 30 yrs ago. The job structures have changed dramatically. He believes that the future of crystallographic computing is probably in the safe hands of a number of crystallographers who increasingly are going to utilize modern computing tools and trends such as open-source collaboration and freely available libraries combined with modern and modular programming languages. I suspect that that some of this is seen differently by a well funded lab in Germany and not everyone would agree that the expertise is being transferred by sufficient training, except in a very few special labs world wide.

David Watkin (Author of CRYSTALS; co-investigator for CCP14)

David reported a little less positive outlook, recapping on the many programs that had existed (>20-30 yrs ago) but which had gone to their software graves and said that there would be only one major package remaining after a few more years (others disagreed of course! CJG mentioned SIR, for one such alternative prgm).

David bemoaned the lack of diversity if this were to become the case, ie one giant system only in use, since he believed the subtleties between the details in the programs were vital in promoting new developments, by maintaining competitiveness between the senior authors and their program suites. These comments promoted very lively discussion, not least why many of us continue to maintain and believe in an artificial divide between the small and macro-molecular fields. He painted the future of crystallographic software as a relatively bleak wasteland that can only be saved by a concerted effort of a dedicated group of people who should design a 'new programme' from the bottom up and made three points at the end of his talk. We could, he suggested either (i) leave things as they are ;(ii) accept disintegration of existing programs as a fact of life; or (iii) stand aside and watch/help others create new software, and to make sure the new program somehow contains all current experience and that the minute details are not lost in the shadows. An area of serious concern for David is that there is potential for a total loss of certain 'knowledge', when the authors of the currently used major programme suites are no longer with us. He added that issues to be addressed within any new framework are funding; staffing; management; location, IP on code etc. He presented a written summary before and at the meeting which was very helpful. It is not reproduced here.

Eleanor Dodson (Director of CCP4)

Eleanor described the CCP4 and the way they've been working over the last ~20 years and reported that they have been generally very successful with their 'modular' approach to such an extent that nowadays, usage and problems related to any macromolecular software inevitably mean "CCP4". The project evolved from a very real need at the time by many groups when there was a huge increase in the amount of synchrotron data being recorded from protein crystals and there wasn't the software to handle these data. CCP4 was developed within very clear guidelines and operating principles which have worked well. [common formats, modular approach, open code/open access, good level of documentation from authors to help users, copyrights always flagged and referenced etc.] They are supported by the research councils and also gain funds from selling licences to the pharmaceutical industries. CCLRC at Daresbury maintain and distribute software, provide tutorials etc from core funds and CCP4 additionally organise workshops and these latter are extremely popular and very much a strength of the project. They attract a large number of students every year and many of the PX researchers see this as their main meeting each year.

It has been a very successful project and much of the success is seen by outsiders as due to Eleanor's management thereof and there is some concern about what will happen when she retires later this year. [if indeed she does retire?] CCP4 could be taken as a model for the future, but all aspects could not be transferred simply from macromolecular to small molecule crystallography. Again this pre-supposes some sort of real divide which some of us believe, is changing rapidly.

Ton Spek (Chairman of the IUCr Commission on Crystallographic Computing)

Ton introduced 3 points at the beginning of his presentation, i) maintenance of existing software; ii) development of new software; iii) automation of existing procedures. He gave an overview of the various 'packages' available and their strengths. He was concerned about training aspects for the future generations and mentioned the decline of the highly successful, triennial crystallographic computing summer schools previously linked to the General Assemblies of the IUCr,that had started in the 60's and had almost disappeared by now. He reported that there was to be a serious computing school to be held (Sienna) in connection with IUCr 2005 in Florence. GMS added a few remarks here about the use of

resources in protein field to support many more workshops that go into details of the software; that teach specific topics in depth and he felt his was a very cost effective use of the networking money raised in training future generations.

John Evans (Powder diffraction expert; X-ray and Neutron)

John talked about the powder diffraction aspects of things, summarising the codes that are available now, those in common usage and how he sees the needs changing for future and challenging applications, regarding flexibility as one of the major requirements of new code /packages: applications include-- high through put data collections on new synchrotrons, the need to analyse sequentially, and simultaneously, 100's and even 1000's of data sets; to include symmetry codes for difficult problems like magnetic structures, for multiphase and multi-source analyses and to have improved indexing routines, more automation and hence serious validation steps. John was saying that TOPAS seems to be capable of providing much of what is needed for work with powder diffraction data in the future because of its approach, code structure and flexibility.

There was further discussion at this point about dividing Lines between Macromolecular Crystallography, Single Crystal Crystallography and Powder Diffraction, with some very polarised views.

Bob Cernik (CCLRC and Manchester; previous Chair of CCP14)

Bob talked about the CCP's in general from historical viewpoint setting the scene and how in particular he saw the development of CCP14 and CCP4 into the future. He sees the future of software development in the small molecule and powder areas under the roof of the CCP14, or some new 'extension' of CCP14. In this respect Bob advocated split funding between CCLRC and other research councils for the support. He elaborated upon this theme and also reported that CCP14 will continue at present funding level with one PDRA.

Kevin Cowtan (Author of the CLIPPER library)

Kevin introduced the topics of the cctbx and the clipper library to the group. His talk was a coherent contribution in terms of the basics of the problem, describing the languages past and present and the reasons for selecting the C++ /python route as a modern object-oriented language of considerable flexibility and strength. Kevin described his own clipper library and referred to the cctbx consortia and the related [commercial] PHENIX system. These research developments are now NIH funded within the Structural Genomics initiatives in the States and progress has been much more rapid recently as a result. GMS was well aware of the status of these developments/programs and had worked with some of the consortia. It was interesting that amongst the number of people present, several had not heard about the new developments.

Rob Hooft (Bruker -Nonius)

Rob implied that the instrument manufacturers aren't really that interested in the whole open software side of things. Their job is to create instruments and to maintain the best data collection /data reduction software for their machines. The CCP4 has shown that perhaps the manufacturers really don't need to be doing this sort of thing, since synchrotron data do not derive from one specific instrument but from a wide range of home-grown machines on different sources. On the other hand, we know that in the small molecule market the software tools provided by the manufacturer can make a huge difference as to which machine the user is going to buy. The manufacturers are very keen to keep a watching brief on discussions such as this present workshop, hence our invitation for them to attend and their genuine keenness to be there.

The meeting ended about 5pm as people left for the airport and trains. It seemed that everyone had an opportunity to contribute and to join the very open discussion during the day and we hope to follow up this with an informal get-together at the ECM meeting in Budapest at the end of August.

Final comments from Horst/Luc

"I believe that recent advances in computing have made software development much more modular and accessible to a much wider range of people than ever before. The cctbx is poised to be the de-facto source of crystallographic computing modules. The project has guaranteed long-term funding and has proven to be very successful - in the macromolecular crystallography world. I don't believe in any dividing line between the different areas of diffraction physics. Different tools are needed in different areas, and that's exactly what a highly modular framework can accommodate best. I believe that ALL crystallographic computing tools comprising ALL areas can be - and inevitably will be - incorporated in the cctbx.

I think that this incorporation process can be somewhat guided and overseen by a dedicated, small group of small molecule and powder crystallographers. I also can conceive of a dedicated small molecule and/or powder programme to be created along the lines of the Phenix Project for macromolecular diffraction"

JAKH/HP/LB Durham
9 June 2004

Age Concern - the Background : Appendix 4
Age Concern : Crystallographic Software for the Future : Project Aims and Objectives

PROJECT AIMS AND OBJECTIVES

It's evident from the introduction that the need for modern, forward-looking crystallographic software is universal. The new platform will continue to underpin the same science areas that we work in currently, but - more importantly - will enable new areas to develop, expand and excel.

Aims

To prevent loss of knowledge which occurs when authors of significant software retire or leave the field. The authors of almost all the major software items are now into, or past, the last quarter of their careers. Though many no longer actively contribute to programming, we are assured of their help and advice in this new venture.

To avoid future software crises: Not only is this project a collaborative effort of researchers distributed over two sites, but an active participation in the development of code to be integrated is invited and encouraged. This places the creation process of this software into stark contrast with the way crystallographic software used to be written: by highly skilled individuals, whose direct input was - and still is - absolutely required if even the slightest bit of functionality is to be altered or added to existing code. The multi-author nature, together with the high standard of documentation that we will enforce during the creation of the code, will ensure the straightforward maintainability of the code into the future.

To reduce the long-term cost of software maintenance. Developing software within a standard framework reduces dependencies on platform specific libraries since this is abstracted by the framework. Porting the entire system to new computing platforms in the future is then greatly simplified, since only minor modifications will be needed.

To complement the planned development strategies already being funded in macromolecular crystallography, within the domain of small molecule crystallography. Opportunities exist for low-level code-sharing between the two communities but the high level implementational details are quite different (as a simple analogy, imagine private cars all being replace by lorries to save on development costs!).

To ensure the emergence of new science in the area of computational crystallography as a direct result of the availability of the new framework. New ideas, no matter how far off- the-beaten-track, can be designed, implemented and used with very little effort and with no serious concern for any infrastructure needed by the software.

Objectives

We will provide a C++/Python framework which will enable research workers to develop, test and implement new ideas quickly and easily. This will avoid needless reimplementation of standard crystallographic mathematics which are in-place in the framework (e.g. symmetry, structure factors, coordinate transforms). Furthermore, the anticipated widespread use of our modules in the crystallographic community will ensure that their new algorithms will reach a wide audience.

We will provide a unified structure to give crystallographers easy access to a diverse choice of algorithms. This is essential for progress as the thrust of small molecule crystallography moves away from simply determining molecular structures and moves towards understanding wider problems in the solid state. Completely new and currently 'unavailable' (to the inexperienced user) methods for the solution of a variety of problems will become routine. Tools exist in the literature for modelling disorder,

incommensurate structural data, optimizing Z' > 1 structures etc, but these are not available in many popular programs, and their use is far from routine or automated. We will develop a reference application based on this newly created framework. On the one hand, this will serve as a test-application for our own development needs, and on the other hand this will provide to the crystallographic community a fully functional, single crystal refinement application with unprecedented functionality, flexibility, customisability and extensibility.

We will thoroughly document every piece of code to the highest standard.

Small Molecule ToolBox

Luc J. Bourhis, Richard J. Gildea, Oleg V. Dolomanov, Judith A.K. Howard & Horst Puschmann

Department of Chemistry, University Science Laboratories, South Road, Durham, DH1 3LE, United Kingdom. E-mail: luc.bourhis@durham.ac.uk

Place holder page. Article-proper starts on following page.

Small Molecule ToolBox

Luc J. Bourhis Richard J. Gildea Oleg V. Dolomanov
Judith A.K. Howard Horst Puschmann

Department of Chemistry,
Durham University, UK

1 Introduction

This article belongs to a series of three published in this edition of the newsletter. The other two are respectively about the program Olex2 [7], developed by the same Durham group, which provides an point-and-click interface to several of the tools we will expound in this newsletter, and about new advances endeavoured by the Oxford end of our common EPSRC grant. In this article, we will concern ourselves with the contributions made to the `cctbx` [2] with the goal to enable small molecule structure refinement up to the standards required for publication, a goal which has become temptingly close at the time of writing this newsletter.

Although the `cctbx` new developments have increasingly become targeted towards macromolecular works, a consequence of its being the foundation upon which the Phenix [3] suite is built, all the core algorithms are valid for any crystal structure. The most important example is that all algorithms in the `cctbx` module[1] are correct in any setting of the 230 crystallographic space groups[2] and that they are routinely tested with space groups not found in protein crystallography. However, there are key differences between macromolecular and small molecule studies in the way crystal structures are modelled and in the way the fit to the diffraction data is performed, interpreted and reported. As a result, the `cctbx`, as it stood four years ago at the beginning of our grant, could not be used for routine small molecule works. Therefore, on the one hand, we have improved existing or added new tools to the `cctbx` where it seemed they were not small molecule-specific and on the other hand we have spun a sister library off, the Small Molecule Toolbox, or `smtbx`, for the more dedicated tools. In practice that dividing line has not been strongly enforced when the changes to the `cctbx` would have threatened its stability. In those cases, we preferred to roll out a new implementation in the `smtbx` (a key example being structure factor computations).

This newsletter will be organised as follows.

The new ab-initio solution method known as charge flipping [15, 16] has become increasingly popular in recent years for small molecule structures, which motivated us to implement that algorithm in the `smtbx`. We will discuss it in section 2.

The refinement of small molecule single crystal structure has a few key specificities, which reflects in the feature set provided by the programs SHELXL [18] and Crystals [5], with which nearly all traditional[3] small molecule studies are carried out. This will be the subject of section 3 where, after a short reminder of the specificities of small molecule refinement, we will discuss

[1] The reader is referred to the previous `cctbx` newsletter for the general organisation of the toolbox, especially the very first one [8].

[2] E.g. settings with an inversion centre not placed at the origin are allowed, which is not the case in any other small molecule program.

[3] By "traditional", we mean no incommensurate crystals and no charge density models.

in turn the LBFGS refinement scheme, the full matrix scheme, normal equations, the floating origin problem and restraints and constraints.

We will then briefly discuss some of the graphs for the analysis of reflection statistics that have been implemented using the `cctbx`.

2 Charge flipping

The first difficulty to overcome in any practical charge flipping algorithm is the initial guess of the flipping threshold δ: too low and the initial random electron density will never be altered enough to reveal structures, too high and emerging structures will be swamped and not given enough time to build up. Our current implementation relies on the map noise σ: $\delta = k\sigma$ where k is a parameter that can be specified at runtime, with a reasonable default value of 1.1.

The second difficulty to overcome is the detection of the phase transition. Following Gabor Oszlányi's advice, we elected the map skewness as the best indicator, which brutally increases at the phase transition as the tail of high density grid points corresponding to the scatterers emerges from the sea of low density. However, before and after the phase transition, the skewness, or any indicator for that matter, significantly fluctuates, which complicates the detection of the phase transition. We therefore use a smoothing technique known as exponential moving average.

The third and last difficulty is to find the origin shift. Indeed charge flipping works in P1 and the structure is shifted compared to the standard space group settings. SUPERFLIP does actually discover the space group symmetry from the final map along with the origin shift, which is clearly the approach to be preferred since it makes charge flipping an ab-initio method by itself. Our code is much cruder and it relies on an a priori knowledge of the symmetry. Moreover we use the macromolecular technique known as fast translation function to recover the origin shift, a method which scales like the 4th power of the number of symmetry elements, which makes it extremely slow on those highly symmetric space groups common in inorganic materials. We are currently working on improving on this weakness.

Finally, after the solution has been obtained, a few cycles of polishing using low density elimination [19] are performed. This polishing could be performed after any solution method actually and it has proven to be invaluable to get good electron density peak positions from which one can assign atom types from geometric considerations. This is at least the experience accumulated with AutoChem, the automatic system developed by our group for Oxford Diffraction, which is available through Olex2.

The implementation is nearly entirely in Python, and resides in the module `smtbx.ab_initio.charge_flipping`. It features the original method as well as the newer one enhanced to deal with weak reflections [14]. Here is the starting point to explore these tools

```
from smtbx.ab_initio import charge_flipping

# low level iterations over the charge flipping cycles
for cycle in charge_flipping.weak_reflection_improved_iterator(f_obs):
    # this is executed at each cycle and one can inspect the situation:
    rho = cycle.rho_map
    c_tot = rho.c_tot() # total charge
    c_flip = rho.c_flip(flipping.delta) # flipped charge
    skew = rho.skewness()

# higher level interface featuring the δ-guessing and phase transition detection
flipping = charge_flipping.weak_reflection_improved_iterator(f_obs)
solving = charge_flipping.solving_iterator(flipping)
charge_flipping.loop(solving, verbose="highly") # for maximum printout
```

3 Refinement

3.1 Introduction

The key specificities of small molecule refinement compared to macromolecular work are threefold:

1. the systematic use of so-called full matrix least-squares;
2. the use of weighting schemes;
3. the mundane use of constraints.

Full matrix least-squares is deemed necessary in order to get the variance-covariance matrix for the parameters of the atomic model which has been fitted to the diffraction data. Then derived quantities such as bond lengths, angles, etc can be computed with estimated standard deviations (e.s.d) which eventually depend on the σ's quoted for each value of $F_o(h)$ or $F_o^2(h)$. This has deep and wide reaching consequences on the possible organisation of numerical code. In this context, the fastest method, if not the most robust, to solve the least-squares minimisation problem is by the method of normal equations.

Weighting schemes work by replacing each aforementioned experimental σ by a function of $F_o(h)$ and $F_c(h)$ which is tuned so that the resulting weighted residuals do not show any trend with the magnitude of $F_c(h)^2$ or $F_c(h)$ for respectively F^2 and F refinement or with resolution. This fine tuning is only performed in the very last stages of the refinement.

Constraints are used even in trouble-free structures, to idealise the geometry of hydrogen atoms, so as to significantly increase the ratio of reflections to parameters, for which a lower bound is required for publication, as enforced by the CHECKCIF system[4]. They are also necessary to model disorder, in order, at least, to constrain the occupancies of corresponding disordered parts to add up to unity, but sometimes also to stabilise the refinement by forcing ADPs to be exactly equal. Constraints need to be taken into account at three stages of the refinement: in the computation of the derivatives at the beginning of each refinement cycle, in the application of the shifts at the end of each refinement cycle and then in the computation of the variance-covariance matrix.

3.2 Overview of a typical macromolecular cycle

We will first introduce the scheme used for macromolecular refinement which is pervasive throughout the entire `mmtbx` and `cctbx` module too. We denote by L the minimisation target, e.g. the least-squares

$$L = \sum_{\text{Miller indices } h} w_h \left(F_o(h) - K\left|F_c(h)\right|\right)^2, \tag{1}$$

or more likely a maximum likelihood estimator, which would be function of the F_c's too. An oversimplified cycle would proceed as follow[5]:

1. compute $F_c(h)$ for each Miller index h, storing those structure factors in an array;
2. compute the minimisation target L for those F_c's as well as the derivatives $\frac{\partial L}{\partial F_c(h)}$ for each h, storing them in another array;

[4] Respectively 10:1 and 8:1 for respectively centrosymmetric and non-centrosymmetric structures

[5] the biggest caveat is that in reality there is no loop over Miller indices since the FFT method is used to compute the structure factors and their derivatives: we chose this unrealistic alternative to ease our presentation and to contrast with the full matrix case in the next section

3. Initialise the gradient ∇L of L with respect to the model parameters to zero; then for each Miller index h:
   (a) compute the gradient $\nabla F_c(h)$ of $F_c(h)$ with respect to the model parameters;
   (b) compute $2\,\mathrm{Re}\left[\frac{\partial L}{\partial F_c(h)}\nabla F_c(h)\right]$ and add it up to ∇L;
4. pass ∇L to the LBFGS minimiser which will compute the parameter shifts to apply for this cycle, as well as internally improve its estimation of the Hessian[6] of L from the last 5 cycles, so that better and better shifts can be computed as cycles pass.

This sequence of operations is typically laid out in Python:

```
# Initialisation before first cycle
from cctbx import xray
target_functor = xray.target_functors.intensity_correlation(f_obs)
structure_factors_functor = xray.structure_factors.from_scatterers(
    miller_set=f_obs)
target_gradients_functor = xray.structure_factors.gradients(miller_set=f_obs)
x = flex.double(number_of_refined_parameters)

# Typical cycle
# step 1.
f_calc = structure_factors_functor(xray_structure, miller_set=f_obs)
# step 2.
target_linearisation_wrt_f_calc = target_functor(f_calc, compute_gradients=True)
# step 3.
grad_f_calc = structure_factor_gradients_functor(
    xray_structure,
    miller_set=f_obs,
    d_target_d_f_calc= target_linearisation_wrt_f_calc.derivatives(),
    n_parameters=x.size()).packed()
```

This scheme is efficient. When it comes to efficiency, three aspects need to be considered: number of floating point operations (flops), the size of the manipulated data and and the size of the machine code which is executed. The latter is often overlooked. First, a bloated library will result in a slower starting time for any program using it, as the library needs to be loaded into memory from disk. Then, one single program may use different features of the `cctbx` at different times and the greater the amount of code to load for each feature, the greater the switching time between parts of the program using those different features, as the new module needs to be loaded from disk and the old one may need to be shelved away to disk as part of the virtual memory system.

Since the loops over the Miller indices are implemented in C++ code which is called by the Python objects `target_functor`, `structure_factors_functor` and `target_gradients_functor`, the computation speed is optimal. Moreover the C++ code for each of those is completely independent from the others. As the result the total code size is the sum of the code size of each needed component, which constitutes the optimal case again. For example, if the program above wishes to use a least-squares target, the memory footprint will be increased only by the code size of that new component.

It should also be noted that weighting schemes and constraints are easy to add into that scheme, keeping its modular nature. The former requires computing F_c-dependent weights between step 1 and 2. The latter requires to alter ∇L after step 3 using the chain rule before passing it to LBFGS and then to alter the shifts after that step. The `cctbx` newsletter '08 demonstrated [6] how the reduction of gradients and expansion of shifts is performed in the case of special position constraints and the reader is referred to it for detailed explanations.

[6] i.e. the matrix of second-order derivatives

3.3 Full matrix cycle

First a mathematical remark is in order. The non-linear least-squares method discussed in this section does not require the computation of the second-order derivatives of $F_c(h)$ any more than the method described in the previous section. But instead of relying on LBFGS to estimate those second-order derivatives based on some general heuristic, one can take advantage of the special structure of the least-squares target to compute a special approximation of those second-order derivatives. That method is centuries-old as it was known to Gauss, and it is actually often named after him: the Gauss-Newton method. The normal equations are just one particularly efficient implementation of that method. Compared to LBFGS, each cycle is more costly but makes greater progresses toward the minimum.

The traditional implementation, as found in existing FORTRAN codes, follows the scheme below. We will denote by x the crystallographic parameters of the model (sites, ADPs, etc) and by y the set of independent parameters after taking constraints into account (independent sites, independent ADPs but also extra parameters as needed by some constraints as exemplified later).

1. Initialise the normal matrix A_y and the right hand side b_y of the normal equations to zero.
2. For each Miller index h:
   (a) compute $F_c(h)$ and $\nabla_x F_c(h)$;
   (b) compute $\nabla_x \left|F_c(h)\right|^2 = 2\,\mathrm{Re}\left[F_c^*(h)\nabla_x F_c(h)\right]$;
   (c) compute $\nabla_y \left|F_c(h)\right|^2$ by using the constraint matrix $C = \left[\frac{\partial x_i}{\partial y_j}\right]_{ij}$: $\nabla_y \left|F_c(h)\right|^2 = C^T \nabla_x \left|F_c(h)\right|^2$;
   (d) compute the F_c-dependent weight $w(h)$ as per the weighting scheme;
   (e) compute the least-squares residual $r(h) = w(h)\left(F_o^2(h) - K\left|F_c(h)\right|^2\right)^2$;
   (f) form the rank-1 matrix $\left[\nabla_y \left|F_c(h)\right|^2\right]\left[\nabla_y \left|F_c(h)\right|^2\right]^T$ and the vector $r(h)\nabla_y \left|F_c(h)\right|^2$; add them up respectively to A_y and b_y.
3. Solve $A_y\eta = b_y$ for the shifts η of y and apply them to y to get the new value of the independent parameters.
4. Compute the new value of the crystallographic parameters from the known dependency to the independent ones.

A few comments are in order.

First, at step 2a, one can take advantage of the mathematical form of the structure factors to compute the gradient as a side-product of the computation of $F_c(h)$ with very little overhead. This is to be contrasted to the previous section where the structure factors and their gradients were computed separately by two completely different bodies of code. This is what motivated the module `smtbx.structure_factors.direct.standard_xray`. It features a computation valid in any space-group and one hand-optimised for the case of centrosymmetric structures, for which one can exploit the fact that $F_c(h)$ is real and $\nabla F_c(h)$ is imaginary to save flops.

Secondly, at step 4, it would not have been correct to simply compute the shifts ξ for the crystallographic parameters x by applying the constraint matrix to the shifts η of the independent parameters, $\xi = C\eta$, and then to apply those ξ to x. This would indeed not work for non-linear constraints. For example, a CH_3 group rotating around the $C-C$ bond, would need an azimuthal angle as part of the vector y. After application of the hydrogen site shifts

computed from the shifts of that angle, the geometry would not be that of an ideal tetrahedral CH_3 anymore. The more cycles, the more distortion would result. Thus the exact formula expressing the sites of the hydrogen atoms as a function of the shifted angle shall be used at the end of each cycle to keep the constraint exactly enforced throughout all the cycles, which is the very reason for a constraint to be used in the first place.

Thirdly, compared to the macromolecular scheme, where the work was split into 3 loops over the Miller indices, all the work needs to be done in one and only one loop here. Any splitting would result in a loss of performance. Let us elaborate on this.

- Why not compute the normal matrix A_x for the crystallographic parameters x and then reducing it to A_y with $A_y = C^T A_x C$?
  With the scheme outlined above, the cost of accumulating the normal matrix is $O(mn_y^2)$ where m is the number of reflections and n_y the number of independent parameters. The proposed alternative would make it $O(mn_x^2)$. The ratio n_x/n_y can be as large as 2 in practice, a case which would result in a fourfold increase of computational cost.

- Why not store the design matrix, each row of which is one $\nabla_x |F_c(h)|^2$?
  At the resolutions used for small molecule work, the reflection to parameter ratio m/n_x is routinely around 30. For example, a structure with 4096 parameters and a data to parameter ratio of 32, refined in double precision, results in a design matrix occupying 4 GB whereas the normal matrix occupies just over 64 MB, a dramatic difference indeed.

This scheme is efficient from the point of view of flops and data size but it may be dreadful from the point of view of code size. Indeed, the code for the whole of step 2 must be compiled in one block. If one wishes to keep each component independent from the others, then for each combination of structure factor algorithm, weighting schemes, constraint system and minimisation targets, we need to compile the whole block. Thus, for example, merely adding another weighting scheme will result in duplicating the whole block of machine code instead of augmenting the code size by the weighting scheme code which is typically tiny. As a result, the total code size needed to enable all possible component combinations is not the sum of the code size of each component anymore: it scales as the number of those combinations. This can result in a very significant overhead. The traditional solution in FORTRAN code is to have a monolithic block which does all alternatives and branches depending on flags passed at runtime. But such a design defeats modularity and makes future extension and maintenance difficult, which is precisely one of the reason to have chosen modern programming languages such as C++ and Python in the first place. This is clearly a difficult problem where some tradeoffs will have to be made.

3.4 Normal equations free of the overall scale factor

We have not addressed the issue of the overall scale factor. Indeed the least-squares target

$$L = \sum_h w_h \left(y_o(h) - K y_c(h)\right)^2, \tag{2}$$

where $y_c = |F_c|$ or $|F_c|^2$, is a function of that scale K which is a priori unknown. The traditional solution consists in adding K to the list of refined parameters, and therefore to add a line and a column to the normal equations corresponding to K.

Another solution is to note that L, as a function of K, is a second-order polynomial and therefore that the value K^* realising the minimum has an analytic expression,

$$K^* = \frac{\sum_h w_h y_o(h) y_c(h)}{\sum_h w_h y_c(h)^2}. \tag{3}$$

Obviously, K^* is a function of the crystallographic parameters that $y_c(h)$ is a function of. It is then obvious that minimising $L|_{K=K^*}$ is still a least-squares problem since that last expression is a sum of squares too. The normal equations for the minimisation of $L|_{K=K^*}$ can easily be computed as a correction to the normal equations for the minimisation of $L|_{\text{fixed } K}$. Only second order derivatives of L need to be corrected actually, since by definition of K^*, the first order derivatives are obtained by merely plugging K^* in the generic formula valid for any K. This is implemented in the C++ class `normal_equations_separating_scale_factor` defined in `scitbx/lstbx/normal_equations.h`. This crystallography-agnostic code is in turn used by the `smtbx`, specifically the function `build_normal_equations` defined in `smtbx/refinement/least_squares.h`.

It should be noted that eqn (3) is used in the `cctbx` too but in the context of LBFGS minimisation, in the C++ class `ls_with_scale` defined in `cctbx/xray/targets.h`. Because of the remark just made about first-order derivatives, this choice of K is totally invisible from the point of view of LBFGS.

3.5 Floating origin

In space groups such as $P2$ or Pm, the normal matrix is singular because $|F_c(h)|^2$ is invariant, for any Miller index h, under a global translation of the whole structure along the 2-axis and along the mirror plane respectively. Let us consider the simplest case to ease the exposition: only 3 sites are refined. Then in the space group $P2$, the vector v defined as

$$v^T = \Big[\underbrace{0\ \ 1\ \ 0}_{\text{atom 1}}\ \ \underbrace{0\ \ 1\ \ 0}_{\text{atom 2}}\ \ \underbrace{0\ \ 1\ \ 0}_{\text{atom 3}} \Big] \tag{4}$$

is a singular vector for the normal matrix, where for each atom the triplet represent shifts in the x, y and z directions. Our restraint consists in adding to the normal matrix a term

$$\mu v v^T. \tag{5}$$

In this case, this is equivalent to restraining the coordinate of the barycentre of the structure along the floating direction to be zero.

The weight μ is chosen as 1000 times the biggest element on the normal matrix diagonal. In fact, the same shifts are obtained for any value of that weight, as long as it is big enough to prevent catastrophic round-offs by keeping the matrix far from being singular. After adding the restraint, the normal matrix is indeed non-singular and the normal equations can be solved. The obtained shift ξ has the property that it has no component along the singular vector v. Thus in this case, that restraint acts as an exact constraint in the sense that the barycentre of the structure will not move. This is actually correct as long as there is no special position in the structure.

Let us consider the space group $R3(-y+z, x+z, -x+y+z)$ (Hall symbol) to illustrate the effect that special positions may have. The 3-axis is in the direction $(1,1,1)$. In general, the singular vector is

$$v^T = \Big[\underbrace{1\ \ 1\ \ 1}_{\text{atom 1}}\ \ \underbrace{1\ \ 1\ \ 1}_{\text{atom 2}}\ \ \underbrace{1\ \ 1\ \ 1}_{\text{atom 3}} \Big] \tag{6}$$

If e.g. the 1st atom is on that 3-axis, its site being then constrained to have the form (x_1, x_1, x_1), the singular vector of the normal matrix reduced by the special position constraints becomes

$$v'^T = \Big[\underbrace{1}_{\text{atom 1}}\ \ \underbrace{1\ \ 1\ \ 1}_{\text{atom 2}}\ \ \underbrace{1\ \ 1\ \ 1}_{\text{atom 3}} \Big] \tag{7}$$

and it is easily seen that adding $\mu v' v'^T$ to the normal matrix still allows a shift of the barycentre. However the property that the total shift vector for the independent site coordinates,

$$\xi^T = \begin{bmatrix} \Delta x_1 & \Delta x_2 & \Delta y_2 & \Delta z_2 & \Delta x_3 & \Delta y_3 & \Delta z_3 \end{bmatrix}, \tag{8}$$

does not have any component along v' holds as in the first case without special positions: this is a general property of the method.

We shall stress that the fact that the barycentre is not always restrained to stay at the origin is not a problem at all. Since the shifts along the singular direction(s) of the normal matrix are null in this scheme, one obtains the so-called minimum norm solution of the least-squares problems. A corollary is that the inverse of the restrained normal matrix is the pseudo-inverse of the original singular normal matrix, which is precisely the one with the right statistical properties, i.e. the one which gives the correct variances and correlations under the Gauss-Markov theorem, as any good treaty on Statistics would expound, see e.g. [20].

This flavour of floating origin restraints is implemented in the C++ class `floating_origin_restraints` defined in `smtbx/refinement/least_squares.h`.

3.6 Restraints

As described in a previous `cctbx` newsletter [9], geometry restraints of the kind that are extensively used in macromolecular crystallography have already been implemented in the `cctbx`. However, with the exception of the bond distance restraint, these were not able to deal with symmetry equivalent atoms, as is allowed by most common small molecule refinement programs. These have now been extended to allow symmetry equivalent atoms, and a new bond similarity restraint has been added.

We have also implemented several common restraints on anisotropic displacement parameters (ADPs), including restraints based on Hirshfeld's "rigid-bond" test [11], similarity restraints and isotropic restraints. There appears to be little in the literature with regard to the mathematical details required for the actual implementation of restraints on ADPs in refinement programs. It was therefore necessary to devise our formulae for the equations of restraints and derive their gradients with respect to the least squares parameters. All the residuals we have implemented possess a property we deemed important: they are rotationally invariant, *i.e.* they are left unchanged if the ADP is transformed by any rotation, or equivalently they are frame-invariant, *i.e.* they are unaffected if the frame to which they are referred is rotated.

An extensive set of tests has been written. The correctness of the analytical gradients was confirmed by testing against gradients determined by the finite differences method. The frame-invariance was also systematically tested.

3.6.1 Rigid-bond restraint

In a "rigid-bond" restraint the components of the anisotropic displacement parameters of two atoms in the direction of the vector connecting those two atoms are restrained to be equal. This corresponds to Hirshfeld's "rigid-bond" test [11] for testing whether anisotropic displacement parameters are physically reasonable (see SHELX manual, DELU restraint [17])and is in general appropriate for bonded and 1,3-separated pairs of atoms and should hold true for most covalently bonded systems. We therefore minimise the mean square displacement of the atoms in the direction of the bond. This is equivalent to the formula given by Hendrickson and Konnert [10], with $\theta = 0$.

The weighted least squares residual is then

$$R = w(z^2_{A,B} - z^2_{B,A})^2, \tag{9}$$

where in the Cartesian coordinate system the mean square displacement of atom A along the vector $\overrightarrow{AB}$, $z^2_{A,B}$, is given by

$$z^2_{A,B} = \frac{\mathbf{r}^t \mathbf{U}_{cart,A} \mathbf{r}}{\|\mathbf{r}\|^2}, \tag{10}$$

where

$$\mathbf{r} = \begin{pmatrix} x_A - x_B \\ y_A - y_B \\ z_A - z_B \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \tag{11}$$

$\mathbf{r}^t$ is the transpose of $\mathbf{r}$ (*i.e.* a row vector) and $\|\mathbf{r}\|$ is the length of the vector $\overrightarrow{AB}$.

The derivative of the residual with respect to an element of $\mathbf{U}_{cart,A}$, $U_{A,ij}$ is given by (using the chain rule)

$$\frac{\partial R}{\partial U_{A,ij}} = 2w(z^2_{A,B} - z^2_{B,A}) \frac{\partial z^2_{A,B}}{\partial U_{A,ij}} \tag{12}$$

with

$$\frac{\partial z^2_{A,B}}{\partial U_{11}} = \frac{x^2}{\|\mathbf{r}\|^2}, \qquad \frac{\partial z^2_{A,B}}{\partial U_{22}} = \frac{y^2}{\|\mathbf{r}\|^2}, \qquad \frac{\partial z^2_{A,B}}{\partial U_{33}} = \frac{z^2}{\|\mathbf{r}\|^2}, \tag{13}$$

and

$$\frac{\partial z^2_{A,B}}{\partial U_{12}} = \frac{2xy}{\|\mathbf{r}\|^2}, \qquad \frac{\partial z^2_{A,B}}{\partial U_{13}} = \frac{2xz}{\|\mathbf{r}\|^2}, \qquad \frac{\partial z^2_{A,B}}{\partial U_{23}} = \frac{2yz}{\|\mathbf{r}\|^2}. \tag{14}$$

From inspection of their analytical form, and making the assumption that the magnitudes of $\mathbf{U}_{cart,ij}$ will be much smaller than the values for x, y, z, it can be assumed that the gradients of the residual with respect to the sites will be negligible. Analysis of the gradients computed by the finite differences method has confirmed that the gradients with respect to the sites are typically one order of magnitude smaller than the gradients with respect to $\mathbf{U}_{cart,ij}$, and hence their contribution can be neglected.

3.6.2 ADP similarity restraint

In the similarity restraint, the anisotropic displacement parameters of two atoms are restrained to have the same U_{ij} components. Since this is only a rough approximation to reality, this restraint should be given a smaller weight in the least squares minimisation than for a rigid-bond restraint, and is suitable for use in larger structures with a poor data to parameter ratio. Applied correctly, this restraint permits a gradual increase and change in direction of the anisotropic displacement parameters along a molecular side-chain [17]. This is equivalent to a SHELXL SIMU restraint [17]. The weighted least squares residual is defined as the square of the Frobenius norm of the matrix of deltas, which, after taking into account the symmetric nature of $\mathbf{U}$, can be written as

$$R = w \left(\sum_{i=1}^{3} (U_{A,ii} - U_{B,ii})^2 + 2 \sum_{i<j} (U_{A,ij} - U_{B,ij})^2 \right). \tag{15}$$

The factor 2 in front of the off-diagonal contribution is necessary to make R independent of its orientation with respect to Cartesian axes.

3.6.3 Isotropic ADP restraint

Here we minimise the difference between the Cartesian ADPs, $\mathbf{U}_{cart}$, and the isotropic equivalent, $\mathbf{U}_{eq}$. The weighted least squares residual then reads

$$R = w\left(\sum_{i=1}^{3}(U_{ii} - U_{eq,ii})^2 + 2\sum_{i<j}(U_{ij} - U_{eq,ij})^2\right), \tag{16}$$

where

$$\mathbf{U}_{eq} = \begin{pmatrix} U_{iso} & 0 & 0 \\ 0 & U_{iso} & 0 \\ 0 & 0 & U_{iso} \end{pmatrix}, \tag{17}$$

and

$$U_{iso} = \tfrac{1}{3}\mathrm{tr}(\mathbf{U}_{cart}). \tag{18}$$

The factor 2 is again essential for frame-independence.

3.6.4 Implementation

The implementation of the geometry restraints has previously been described by Grosse-Kunstleve et al. [9], and the ADP restraints were designed in the same way as the pre-existing geometry restraints classes.

The SHELXL SIMU, ISOR and DELU instructions for restraints on anisotropic displacement parameters automatically set up the appropriate restraints for adjacent pairs of atoms (and 1,3- pairs in the case of DELU), using the atomic connectivity table [17]. This can be done for all atoms in the structure, current residue, or given list of atoms. A python class was implemented to emulate each of these SHELXL instructions and create the appropriate shared proxy arrays for each restraint type. These were tested and compared against structures using SHELXL to confirm that both had setup the same restraints.

It was necessary to add the ability to create the `cctbx` atomic connectivity table taking into account the covalent radii of the atoms when deciding whether any two atoms are bonded or not. Previously it was only possible to discriminate bonded from non-bonded by means of a distance cutoff value.

3.6.5 Parsing of SHELXL restraints

Parsing of a crystal structure from a SHELXL instruction file exists in the `iotbx` (input/output toolbox) module of the `cctbx` in the form of a lexer/parser/builder combination. This has been extended to include parsing of SHELXL restraints and building of the appropriate shared proxy arrays required by the refinement.

3.7 Constraints

We will finish this overview of refinement with a quick word on constraints.

Special position constraints are automatically handled, both for sites and ADPs, for LBFGS and full matrix refinement. The core numerical code was discussed in last year's `cctbx` newsletter.

The chain rule has been implemented for all hydrogen geometries supported by SHELXL, including rotating CH_3 groups and stretchable $C-H$ bonds. In the former, this results in one azimuthal angle to be added to the independent parameters, whereas an additional bond length appears in the latter. The implementation has been validated with a comprehensive list of tests. It sits in `smtbx/refinement/constraints/geometric_hydrogen.h`. However it has

not been plugged into either the LBFGS or the full matrix refinement engine yet. The design of the interface for the constraint system is still in a state of flux and this is clearly the next priority.

4 Analysis of reflection statistics

An example of the ease of rapid prototyping with the Python language can be found in the statistical graphs that have been added to the `cctbx` and which are now available from Olex2 [7]. These include Wilson plots, systematic absences intensity distribution, completeness plots and a cumulative intensity distribution. In each cases, it was possible to take advantage of tools already in the `cctbx` such as those for dealing with miller arrays of reflection data, binning of data and general scientific and mathematical tools available in the `scitbx` module. A simple implementation of a systematic absences intensity distribution could be written in just seven lines of Python.

```
class sys_absent_intensity_distribution:
    # I/sigma(I) vs I
    def __init__(self, f_obs):
        sys_absences=f_obs.select_sys_absent()
        assert sys_absences.sigmas() is not None
        intensities=sys_absences.as_intensity_array()
        self.x=(intensities/sys_absences.sigmas()).data()
        self.y=intensities.data()
```

The computation that occurs in the penultimate line

```
intensities/sys_absences.sigmas()
```

actually occurs in C++ code, despite being called in Python.

A class to calculate the Wilson plot was already in the `cctbx`, however this did not calculate the normalised amplitudes or E statistics that are used in space group determination. Therefore, a C++ class was written to calculate the normalised amplitudes and E statistics that were missing from the current python class.

A cumulative intensity distribution class [13] was initially written exclusively in Python, but the core part of the algorithm was rewritten in C++ for increased speed in the computationally intensive loop over potentially as many as several tens of thousands of reflections.

We have recently exposed some of the statistical distributions that are part of the BOOST C++ library [1] to python, in order to enable the calculation of quantile-quantile plots such as normal probability plots [4], or probability plots based on Student's t-distribution [12].

Some small tools were added to the `iotbx` module of the `cctbx` to enable simple outputting of data to a comma-separated values (CSV) file for easy importing of the data into spreadsheet software if required, or the XY data can be passed to a program such as Olex2 [7] for graphical output.

5 Future works

We will conclude this newsletter by laying out the plan for the foreseeable future. As pointed out in several places, there are some clear gaps in our framework which prevent us from using it for crystal structure publications. After implementing full matrix refinement, we have come much closer to that goal since the variance-covariance matrix can now be obtained. We have to finalise the code to do so, which in turn depends on finalising the design of the constraint system and then plug in the most common restraints, geometric hydrogen constraints being the highest priority. At the other end of the workflow, some polishing of our charge flipping code needs to be carried out, especially the efficient recovery of the origin shift and space group.

Acknowledgements

We gratefully acknowledge the financial support of the Engineering and Physical Sciences Research Council (UK) under grant number EP/C536274/1. We would like to thank David Watkin who was key to the birth of this project and who has shared his in-depth knowledge of crystallography during all those years. One of the authors, Luc Bourhis, would like to warmly thank Gabor Oszlányi for openly sharing with him his mastery of the charge flipping algorithm at the Kyoto Computing School. Finally we are all indebted to Ralf W. Grosse-Kunstleve for his infallible support and patience.

References

[1] BOOST C++ libraries.

[2] URL http://cctbx.sourceforge.net.

[3] URL http://www.phenix-online.org/.

[4] S.C. Abrahams and E.T. Keve. Normal probability plot analysis of error in measured and derived quantities and standard deviations. *Acta Cryst. A*, 27:157–165, 1971.

[5] P.W. Betteridge, J.R. Carruthers, R.I. Cooper, K. Prout, and D.J. Watkin. CRYSTALS version 12: software for guided crystal structure analysis. *J. Appl. Cryst.*, 36:1487, 2003.

[6] L.J. Bourhis, R.W. Grosse-Kunstleve, and P.D. Adams. cctbx news. *Comp. Comm. Newsletter*, 8, 2007.

[7] O.V. Dolomanov, L.J. Bourhis, R.J. Gildea, J.A.K. Howard, and H. Puschmann. Olex2: a complete structure solution, refinement and analysis program. *J. Appl. Cryst.*, 42:339–341, 2009.

[8] R.W. Grosse-Kunstleve and P.D. Adams. State of the toolbox: an overview of the computational crystallography toolbox (cctbx). *Comp. Comm. Newsletter*, 1, 2003.

[9] R.W. Grosse-Kunstleve, P.V. Afonine, and P.D. Adams. cctbx news: Geometry restraints and other new features. *Newsletter of the IUCr Commission on Crystallographic Computing*, 4:19–36, 2004.

[10] W.A. Hendrickson and J.H. Konnert. Incorporation of stereochemical information into crystallographic refinement. In R. Diamond, S. Ramaseshan, and D. Venkatesan, editors, *Computing in Crystallography*, pages 13.01–13.26, 1980.

[11] F.L. Hirshfeld. Can X-ray data distinguish bonding effects from vibrational smearing? *Acta Cryst. A*, 32:239–244, 1976.

[12] R.W.W Hooft, L.H. Straver, and A.L. Spek. Probability plots based on student's t-distribution. *Acta Cryst. A*, 65:319–321, 2009.

[13] E.R. Howells, D.C. Phillips, and D. Rogers. The probability distribution of X-ray intensities. ii. experimental investigation and the X-ray detection of centres of symmetry. *Acta Cryst.*, 3:210–214, 1950.

[14] G. Oszlányi and A. Sütő. Ab initio structure solution by charge flipping. ii. use of weak reflections. *Acta Cryst. A*, 61:147, 2004.

[15] G. Oszlányi and A. Sütő. The charge flipping algorithm. *Acta Cryst. A*, 64:123–134, 2008.

[16] L. Palatinus and G. Chapuis. SUPERFLIP – a computer program for the solution of crystal structures by charge flipping in arbitrary dimensions. *J. Appl. Cryst.*, 40:786–790, 2007.

[17] G.M. Sheldrick. *The SHELX-97 Manual*. Dept. of Structural Chemistry, Univ. of Göttingen, Göttingen, Germany, 1997.

[18] G.M. Sheldrick. A short history of SHELX. *Acta Cryst. A*, 64:112 – 122, 2008.

[19] M. Shiono and M.M. Woolfson. Direct-space methods in phase extension and phase determination. i. low-density elimination. *Acta Cryst. A*, 48:451–456, 1992.

[20] S.D. Silvey. *Statistical Inference*, volume 7 of *Monographs on Statistics and Applied Probability*. Chapman & Hall/CRC, 1975.

Small Molecule Crystallography Toolkit

Mustapha Sadki
Chemical Crystallography Laboratory, University of Oxford, South Parks Road, OX1 3QR, United Kingdom. E-mail: mustapha.sadki@chem.ox.ac.uk

Introduction

One of the requirements for the next generation of small molecule crystallographers is a mathematical programming infrastructure, which offers a modelling design, is able to support the whole crystallographic modelling *life-cycle* and keeps model formulations separate from the optimisation process. It should permit the investigation of solver performance and sensitivity analysis for ill-conditioned problems, where one model formulation can be processed with numerous solvers, each of which implements one or more optimisation algorithms.

To this end, we have defined a strategy and designed a Small Molecule Crystallography Toolkit library. This toolkit enables the application of optimisation components in general and refinement-based applications in particular, to crystallographic problems. We have used the concept of a modelling environment, which consists of objective and constraint expressions, a concept that has become an essential tool for a wide range of optimisation and related problems.

In practice, the toolkit provides users with an easy and efficient means to test ideas, construct new algorithms, as well as build large and maintainable models which can be readily adapted to any new situation. This enables users to develop and explore the full capabilities of crystallography, and from which other researchers can create new applications.

C++ integrated modelling environment vs modelling language

During the last two decades, there have been significant algorithmic advances in optimisation problems and valuable developments in software tools for solving linear and non-linear problems. The process of formulating problems in intuitive terms and invoking the software has been simplified with the advent in high-level modelling languages.

Nevertheless, if these algebraic modelling languages represent the strengths, they also have some weaknesses, which can make them a second choice in certain specific applications. These modelling languages offer syntax close to the notation used by most modellers and provide representations readable by both humans and computers. They also generate model-data in an automatic way and simplify the problems of modification, adaptation to a new situation and maintenance. However, these high-level languages also have a major flaw, which can be summarised as follows: they are limited in procedural support for algorithmic development and are difficult to integrate with other software components.

The usefulness of any code may be misjudged by the user if it does not offer a practical and easy interface in applications (even if it provides an effective outcome). The most suitable interface depends strongly on the particular application. In some disciplines, application-specific modelling languages allow problems to be posed in a thoroughly intuitive way; whereas in other disciplines, application-specific graphical user interfaces may be considered more appropriate

Bearing this in mind, a new toolkit is being developed to include common modelling constructs and patterns, whilst also addressing new requirements in the crystallographic domain, which from the outset was our main concern. It allows problems to be specified and maintained in intuitive terms by using ordinary algebraic notation, which helps the crystallographer specify the function whose parameters are being estimated, with or without restraints and constraints.

To take advantage of all the strengths that modelling languages provide while avoiding their drawbacks, we chose to implement this toolkit as an integrated modelling language within C++, so as to combine the modelling facilities with a powerful object-oriented programming language. The toolkit includes a natural crystallographic modelling language together with an interactive command environment, which contains the main patterns needed to formulate new problems, and which was originally used for testing and development.

The underlying concept is that the design of a special-purpose crystallographic language enables the advanced user to specify any kind of relationship between the conventional crystallographic variables and any novel ones that they need to introduce, without any syntactic burden imposed on users, as it is the case for general-purpose programming languages. The users express objectives, constraints and restraints in a mathematical notation, without indicating anything about the partial derivatives that a solver(s) might need. The system deduces "active" variables behind the scenes and arranges automatic differentiation computations, where appropriate.

Nonlinear Programming

Nonlinear programming problems are constrained optimisation problems with nonlinear objective and/or constraint functions, where we assume that all functions in question are smooth (typically, at least twice differentiable):

$$\begin{aligned} &\min \quad f(x) \\ &\textit{subject to} \\ &c(x) \le 0 \end{aligned}$$

where:

- f(x) is the objective function and for nonlinear regression in our domain set to be:

$$M = \sum_h w_h \left(|F_{o,h}|^2 - K^2 |F_{c,h}|^2 \right)^2 + \sum_q w_q \left(g_{o,q} - g_q \right)^2$$

$|F_{o,h}|^2$ is the observed intensity,
$|F_{c,h}|^2$ is the calculated intensity and w_h is the weight to be assigned to each of them. The second part stands for various geometrical or chemical restraints whenever necessary, w_g the weight for the restraint $g_{o,q}$

- c(x) are general nonlinear constraint functions.

Fig.1: *NLS Model Formulation Refinement*

Fig.1 represents the targeted form in this work and provides a solution to such problems. In addition, *sensitivity analysis* has been included as a tool which emerges as a favourite technique for analysing the outcome of the optimisation process. This tool is used to determine how different values of an independent variable will impact on a particular dependent variable under a given set of assumptions and within specific boundaries that will depend on one or more input variables.

Automatic Differentiation

The computation of derivatives is an essential part of numerical optimisation algorithms, sensitivity analysis and scientific programming. Automatic Differentiation (AD) denotes a set of techniques for computing analytic derivatives accurately and efficiently without hand-coding them. The computation is based on the composition of simple operations (+, *, sin(), log(), etc…) with known derivatives combined, using the chain rule.

Contrary to the Finite-difference approximation and the symbolic methods with all their inherent drawbacks, there are two variants of automatic differentiation which are more convenient and efficient. Since the first AD publication of Griewank's 1989 survey [1] followed by the proceedings book [2], these techniques have increased in popularity and become an essential tool within scientific computing.

The most intuitive method for computing first derivatives by AD is the ''Forward mode'', which consists of computing the partials of each elementary operation with respect to the independent variables.
If $f = f(a,b)$ depends on the operands a and b and the partials $\partial a/\partial xi$ and $\partial b/\partial xi$ are known, then we can use the chain rule to compute:

$$\partial f/\partial xi = \partial o/\partial a * \partial a/\partial xi + \partial o/\partial b * \partial b/\partial xi.$$

Even though this method is readily extendable for higher derivatives, it has the disadvantage that the evaluation of an expression may take $O(k^2)$ operations, when only $O(k)$ is necessary, and so makes it cumbersome to use.

The alternative method, which is "Backward mode" AD as opposed to the previous method, computes $\nabla f(x)$ by recurring the partials $\partial f/\partial o$ of f (called *adjoints)* with respect to each operation o involved in evaluating $f(x)$. To achieve this, we visit the elementary operations first in ''forward'' order to compute $f(x)$, then in ''reverse'' order to recur the partials. At the end of this ''reverse sweep'', we get $\partial f/\partial xi$, which are the components of $\nabla f(x)$. In contrast with the Forward mode, this method has the advantage of computing both $f(x)$ and $\nabla f(x)$ in at most a small multiple of the time needed to compute $f(x)$ alone; but with the disadvantage that it requires information to be saved for the reverse sweep. Thus it may entail considerably more storage than does the Forward mode AD.

It is good to note that, for solvers/algorithms which require the Hessian, it is conceptually straightforward to extend a backward AD computation of $\nabla f(x)$ to a Hessian-times-vector computation (i.e. $\nabla 2 f(x)\ v$ *for* arbitrary (constant) vectors v); all we need do is apply backward AD to the computation of $v^{\mathrm{T}} \nabla f(x)$ [4]. Bruce Christianson [5] has described another way to handle this computation which consists of using forward AD to compute $\psi(0)$ and $\psi'(0)$, where ψ is a scalar function defined by:

$$\psi : \Re \to \Re$$
$$\psi(\tau) = f(x + \tau v)$$

and then applying backward AD to compute $\nabla^2 f(x)\ v$, which is the gradient with respect to x of $\nabla \psi$.

These AD methods have no restricted limits on the length or the complexity of the code (function) to be differentiated. When applied, computation accuracy has no finite-difference truncation errors [3]. Moreover, when integrated within the optimisation process, they have a beneficial impact by simplifying maintenance and so saving programmer time.

AD implementation

The backward mode AD method was chosen for the implementation, rather than the forward mode, as it has a favourable computational cost and scalability, as required by the targeted domain. The implementation is based on specialised overloading and templating to support different built-in types. To address the drawback of this method related to memory storage and efficiency, the reverse-sweep is implemented to operate on 'tape' memory with a chained block memory allocation structure.

The "active" variables of type smxt::adparam::param_t could simply be assigned values initially, for instance:

smxt::adparam::param_t u = 4; or smxt::adparam::param_t u(4);

and be overwritten when needed. Each operation stores: (values, partials) in memory. Then when the gradient is needed, smxt::adparam::differentiate() is invoked. This method will cause reverse-sweep and reclamation of tape memory for the next call.

Assuming the active variable smxt::adparam::param_t **u** is used, one can get its value by **u**.value() and its derivative with respect to the last computed smxt::adparam::param_t in the model by **u**.adjoint().

Example of use:

Compute derivative of f **w.r.t** x_i

Where: $f(x_0, x_1, ...x_n) = \sum_{i=0}^{n} (\sin(2\pi x_i) e^{(2x_i)})$

using backward AD with reverse sweep:

```
{
        smxt::adparam::param_t     x[n+1], f;       // declare active variables
        // fill x[]  for i=0..n                     //  initialisation
        f = 0;
        for ( int i =0  ; i <= n; i++) {
              f  +=  sin( 2*pi * x[i] )  *  exp( 2 * x[i]  );
        }
        // invoke reverse sweep
        smxt::adparam::differentiate ();

        std::cout  << x[i].value() <<  "\t" << x[i].adjoint() << std::endl;
}
```

Example using the integrated smxt.interpreter

```
{
  parameter   p,p2,q            # declares 3 parameters
  p = 0.2; p2=6                 # initial values
  q = cos  ( 2*p2 ) * exp(p )  # some function

  differentiate()               # invoke reverse sweep
                                # and reclamation of memory
   print "r p : " ,  p, p2      # prints p and p2 as (value, adjoint)
}
```

The same example using C++

```
{   smxt::adparam::param_t   p,p2;
            p = 0.2; p2=6;
            p = cos (2* p2 ) * exp(p );
            smxt::adparam::differentiate();
            std::cout << p << "\t"  << p2 << std::endl;
}
```

AD Test and validation

For the principal crystallographic functions, the AD module was tested and validated using different data-sets with all the analytic derivatives as found in Spagna [6], which for testing purposes were hand-coded separately. They included derivatives for the structure factor and geometrical constraints.

Regarding practical issues, efficiency when coding expressions could be further enhanced by some basic user assistance and good practice. Indeed, it is more efficient to build expressions by modifying existing ones, rather than creating new ones. For instance, the statement ***expr = expr*** + *x* makes a copy of ***expr***, while ***expr*** += *x* just appends a new term. Another example in the statements:

y = cos(x); z = cos(x); the second statement could be replaced by **z = y;**

Parameter Estimation and Refinement

Parameter estimation is a common problem in many areas of process modelling. The goal is to determine the values of model parameters that represent and provide good agreement between predicted and measured data. Methods are generally based on different types of least squares or maximum likelihood criterion. In general, this class of problem is nonlinear and frequently a non-convex optimisation problem.

As a modelling environment for expressing and working with optimisation problems, the proposed toolkit pays special attention to common programming issues for refinement by way of implementing special classes dedicated to this type of problem.

The toolkit aims to:

- provide automatic derivatives and so help towards new algorithm development;
- handle refinement problem-generation and modelling with the objective (residual) and constraints described by algebraic expressions;
- provide an interface between modelling environments and solvers;
- offer built-in crystallographic entities (constraints/restraints...) and algorithms.

This form of integrated modelling language as applied to crystallography from within a high-level language (i.e. C++) intuitively extends and enables the implementation of the most common Structure Factor expressions and constraints.

By using the new environment with the meta-programming and template models for published Structure Factor expressions, we implemented a set of classes for structure factor expressions calculation and refinement. These are built-in classes for refinement with dataset and/or options:

For example:

structure_factor sf<param_x>(Env); instantiate a basic refinement for atom positions and the overall scale-factor.

<param_iso> extends <param_x> to iso parameters refinement
<param_aniso>
<param_aniso_extinction>
<param_aniso_ twinned_structures>
<param_aniso_enantiomorph_ext>
<… and more

To extend and implement other classes for a new Structure Factor algorithm, it is mostly sufficient to overload the residual() member, where the code should only describe and return the underlying algebraic expression, expressed algorithmically. Any additional data member (contained in the created new class) would require an additional overloading of the serialisation operator().

Note that the class for the most general expression of the structure factor is able to achieve all tasks targeted by these partial specialised classes. This is possible with extra settings before and/or during refinement (e.g. fixing/unfixing parameters, drop/restore constraints ...). The provision of such specialised structure factor implementations as built-in classes is justified by the resulting efficiency. Further, it illustrates how this environment makes it easier to implement and extend the SF to more general analytical forms.

Weighting schemes

Many different weighting schemes have been proposed by authors proposing functions based on the observations or a combination of the estimated standard uncertainty σ_h with observations.

It is well known that when the structure attains a satisfactory refinement, the normal procedure is to look at Goodness of Fit (square root of the χ^2 per degrees of freedom).

$$\chi^2 = \frac{\sum_h w_h \left(|F_{o,h}|^2 - \left(K |F_{c,h}| \right)^2 \right)^2}{(n-m)}$$

m and *n* denote the number of refined parameters and the number of reflections respectively.

Where the weight w_h is replaced by a parameterisation $w_h(p_i)$ to depend on certain parameters $p_1, p_2 \ldots$ which are then adjusted to minimise $\left|\chi^2 - 1\right|$. One example among others of these weighting schemes is suggested by Sheldrick and used in the program SHELXL[8]:

$$w_h(a,b,c,d,e,f) = \frac{q(\theta_h, c)}{\sigma^2_{F_{ok}} + (aP_k(f))^2 + bP_k(f) + d + e\sin(\theta_k)}$$

where :

$$P_k(f) = f\max(|F_{o,k}|^2, 0) + (1-f)|F_{c,k}|^2 \; ; \qquad q(\theta_h, c) = \begin{cases} 1 & if\ c = 0 \\ e^{c(\frac{\sin(\theta)}{\lambda})} & if\ c > 0 \\ 1 - e^{c(\frac{\sin(\theta)}{\lambda})^2} & if \quad c < 0 \end{cases}$$

Using this algorithm, a model can be built simply within the proposed environment, by coding the algebraic expression of the objective function, that is $|\chi^2 - 1|$. This procedure thus benefits from the advantages of the solver-independent language offered to achieve such a minimisation. The optimal weight can then be set on runtime to finalise the refinement.

All common formulae which have been used in crystallographic software are integrated as built-in classes and can be set statically or dynamically during refinement. Any new formulae for weighting schemes can be defined, implemented and set dynamically by the user.

Constraints

As we can see in the following examples, the constraints of the problem are integrated within the model, simply by adding them as an algebraic expression constructed with the active related parameters. Note that these expressions may contain multiple terms that involve the same variable. These duplicate terms are merged when creating a constraint from the expression. With regards to atoms in special positions, the built-in solvers use an automatic test and reformulation of the model; whereas interfaced solvers need this information as constraints.

Currently, a comprehensive list of the constraints related to hydrogen atom geometries placement, as found in CRYSTALS and SHELXL, is being implemented as built-in constraints, such as aromatic or tertiary CH, idealised secondary CH2 and terminal CH3 and NH2 groups.

Solvers

In addition to the conventional crystallographic solvers that are built-in, the open architecture we have adopted has enabled some useful external and modern non-linear solvers to be successfully interfaced to the system.

Example of solvers:

- Normal matrix / LU /QR / SVD decomposition, CGradient
- Generalised Minimum RESidual (GMRES)
- Levenberg-Marquardt non-linear minimisation with bounds on the parameters or linear constraints
- LBFGS-b with bounds on the parameters
- and IpOpt with bounds and general constraints
- others will be included as needed.

Conventional methods do not guarantee convergence to the global minimum sought in the parameter estimation problem, hence their unreliability. To investigate this, one can set out to solve these nonlinear least-squares problems expressed in the proposed smxt environment and so transform it into a general minimisation problem.

Examples

The following C++ example illustrates certain functionalities in regards to the optimisation, bounds and constraints. The example builds a model, optimises it and outputs the optimal objective value.

The example optimises the following model:

$$\begin{aligned} \textbf{minimise} \quad & c + a * e^{(c+2*b)} \\ \textbf{subject to} \quad & a + 2 * b \geq 4 \\ & a + c = 1 \\ & 0.0 \leq a \leq 5.0 \\ & -2.0 \leq b \leq 2.0 \\ & -1.0 \leq c \leq 0.5 \end{aligned}$$

```
#include "smxt.h"

Int main(int   argc, char *argv[])
{
 try {
   smxt::SmxtEnv env = smxt::SmxtEnv ();

   smxt::model model = smxt::model(env);
   SmxtVar a,b,c;
   // assign initial values
   // ...

   SmxtVar obj = c + a*exp(2*b+c);

   model.objective(obj);

   // Add bounds
   model.add_bound(a, 0.0, 5.0,    "a");
   model.add_bound(b, -2.0, 2.0,   "b");
   model.add_bound(c, -1.0, 0.5,   "c");

   // Add constraint: a + 2 b  >= 4
   model. add_constr ( a + 2 * b >= 4 , "c1");

   // Add constraint: a + c == 1
   model. add_constr ( a + c == 1, "c2");

   // Optimise model
   model.optimise();

   cout << "a " <<  a.value() << endl;
   cout << "b " <<  b.value() << endl;
   cout << "c " <<  c.value() << endl;

   cout << "Obj: " << model.get_objvalue() << endl;

 } catch(SMXTxception e) {
     cout << "Error code = " << e.get_error_code() << endl;
     cout <<  e.get_message() << endl;
 } catch(...) {
     cout << "Exception during refinement " << endl;
```

```
  }
  return 0;
}
```

Fig. 2: *Example of modelling with smxt*

The first executable statement in the example obtains a smxt environment (using the SmxtE*nv()* constructor):

SmxtE*nv* env = SmxtE*nv* ();

All optimisation models require this environment.

Creating the model

Once an environment has been created, the next step is to create a model. A smxt model holds details for the optimisation problem. It consists of a set of variables, optional set of bounds and an optional set of constraints and the associated attributes (variable bounds, objective function, e.g. residuals, constraint senses, constraint right-hand side values, etc.).

The first step towards building a model that contains all of this information is to create an empty model object:

smxt::model model = smxt::model(env);

Adding bounds to the model

After defining the objective within the model, the next step in the example is to add optional bounds to the model.

Bounds are added through the *add_bound ()* method on the model object. A *bound* is associated with a particular model.

The first argument to the *add_bound ()* call is the algebraic expression form for a constraint. The second and third arguments are lhs and rhs coefficients respectively. The final argument is the name of the bound.

Adding constraints to the model

The next step is to add the constraints. The first constraint for example is added here with the statement:

model. *add_constr* (a + 2 * b >= 4 , "c1");

Constraints are associated with a specific model and are created using the *add_constr()* or *add_constrs ()* methods on the model object.

The constraint is built here using overloaded operators. The arithmetic and comparison operators are all overloaded in the C++ to allow the creation of objects of the appropriate types. A symbolic form is another format in which we can add constraints.

Optimising the model

After building the model, the next step is to optimise it by calling:

```
model.optimise();
```

This routine generates a model-data instance, performs the optimisation and populates internal model attributes: status of the optimisation, the solution, parameters, etc.

Error handling

Errors in the smxt environment are handled through the C++ exception mechanism. In the example, all statements are enclosed inside a *try* block and any associated errors are caught by the associated *catch* block.

Example with simple algebraic notation:

Fit plane to data points

Conceptualisation: Minimise Perpendicular Distance Points to Plane. In a traditional informal algebraic description, the implicit equation for a plane in orthogonal 3D space is:

$$A\,x + b\,y + c\,z + d = 0$$

If c is not 0 (e.g. we do not have a vertical plane) this can be written as:

$$\mathbf{a\,x + b\,y + z + d = 0}$$

The perpendicular distance of a point (x,y,z) to the plane is:

$$\text{Distance} = | a * x + b * y + z + d | \,/\, \text{sqrt}(a^2 + b^2 + 1)$$

The model minimises the sum of the squared distances.

Algebraic implementation with the interpreter

```
observation      x, y, z;     // observations to read from 'experiment.dat' file
parameters       a,  b, d;

//minimise  sum {  all observations } ( obs - calc)^2
 residual:
                 abs( a * x + b * y + z + d )/ sqrt(a^2 + b^2 + 1);
                 // reads data points x y z from 'experiment.dat' file
 data            "experiment.dat";

 // then we call for refinement
 refine
 print a,b,d;
```

Implementation in C++ using the smxt library:

//All we need to do is to define the template function to be used in least square function

```
template<class num_t>
 num_t Distance2Plane ( Array2D<num_t> &p , Array2D<> &data)
{
   return abs(p[0]* data[0] +  p[1]* data[1] + data[2]  + p[2]) /
                    sqrt( p[0]*p[0] + p[1]*p[1] +1);
}
```

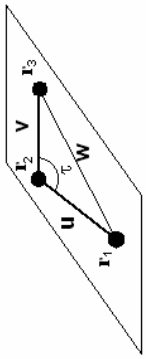
// instantiate the least square object using the template function with optional arguments:

```
    bool  need_covar = true;
   lm_solver<Distance2Plane> lsq(data_points, m, n, need_covar);
   // generates the lsq model f & f(x) and then we call for minimisation
   ret = lsq.minimise (Observation);
```

Example for bond angle restraint application

Having defined the Inter-atomic vectors for 3 points coordinates, r1, r2 and r3

$$\mathbf{u}^i = \mathbf{r_1} - \mathbf{r_2} \rightarrow [(x_1 - x_2),(y_1 - y_2),(z_1 - z_2)]$$
$$\mathbf{v}^i = \mathbf{r_3} - \mathbf{r_2} \rightarrow [(x_3 - x_2),(y_3 - y_2),(z_3 - z_2)]$$
$$(\mathbf{u} - \mathbf{v})^i = \mathbf{w}^i = \mathbf{r_1} - \mathbf{r_3} \rightarrow [(x_1 - x_3),(y_1 - y_3),(z_1 - z_3)]$$



The bond angle $\boldsymbol{\tau}$ at the position r2 is defined by[6]:

$$\cos\boldsymbol{\tau} = \frac{|\mathbf{u}|^2 + |\mathbf{v}|^2 - |\mathbf{w}|^2}{2|\mathbf{u}||\mathbf{v}|} \qquad \text{Equ.1.}$$

where $|u|^2 = \mathbf{u}^i \bullet \mathbf{G}_{ij}\mathbf{u}^j$ and $\mathbf{G}_{ij}$ is direct metric tensor

For the refinement, the implementation of the Equ.1 as bond angle restraint for a known angle will need only the code for the equation as a templated function such as:

```
template< typename  T>
T bond_angle (Tensor &Gij , Coord3d<T> &u,  Coord3d <T>&v , Coord3d <T>&w)
{
    Vector3d<T> u(r1-r2),
    Vector3d<T> v(r3-r2);
    Vector3d<T> w(r1-r3);
     T u2 = u * Gij * u;  T v2 =...

    return  acos(   (u2 + v2 - w2 )   /  ( 2 * sqrt(u2)  * sqrt( v2) );
}
```

The refinement process will then be able to include it as restraint if required and will also use Propagation of Errors to produce the estimated standard deviations for it.

$$\sigma^2(f) = \sum_{i,j=0} \frac{\partial f}{\partial x_i} \frac{\partial f}{\partial x_j} . \mathrm{cov}(x_i, x_j)$$

Propagation of Errors for a function *f(x0, x2, ..., xn);*

Structure Factor and Refinement

As we indicated previously, for all the published common structure factor expressions, special classes are implemented without further need to redefine their residual () member - this only applies if a change has to be made to the related formulae - and one can use the selected class to do the desired refinement.

SF least square snippet simple code in C++:

```
smxt::io:cif::CifReader cif( ciffilename );
smxt::ScattererList Atoms(cif) ;              //   + optional Atoms settings, filter ...
// instantiate the SFLS class using the class param_anisotropic_anamalous<>
// and read any extra  param from cif, e-g :OverAllscalef,  Flack_param …

smxt::sfls:: SfLs< param_anisotropic_anamalous<> > sfls(Atoms, cif);

Reflection hkl(hklfile);                  //  + optional hkl settings ...

sfls.fix_scalefactor();
//  Special postions and adp beta-restrictions constraints are handled internally
sfls.refine_positions();                  // all positions
sfls.refine_uij();                        // all adps
sfls.refine_biso("C_3");                  // refine atom C3 as isotropic

sfls.refine( hkl );
```

The constraints can be set in this process relying on a naming service managed by the SFLS class, which gives predefined names for all parameters and implements a symbolic expression parser. The user can refer to this list to set constraints, fix or unfix parameters (e.g. sfls.refine_biso("C_3")). However, if the user re-implements the structure factor definition, this will provide a method, if necessary, to set constraints internally, as shown in Figure above.

Discussion and Conclusion

A toolkit for practical modelling and solving is described, which helps the crystallographer explore different models as expressed at a high-level of abstraction. Currently, the implementation of new classes is ongoing and we expect to provide different models found in the literature as a built-in, such as a hindred rotator. This environment allows for the extension of many models to their general form.

Moreover, this toolkit is designed to facilitate the solution of nonlinear least squares and support the whole crystallographic modelling life-cycle*:* building, refining, analysing and revising. Furthermore, the design of the proposed toolkit for modelling and interacting with solvers allows for:

- hybrid refinement with the combination of more than one model and more than one solver;

- powerful algorithmic procedures by alternating between models. Two (or more) models are solved in alternation, where an optimal solution for one model yields new data for the other;

- taking care of common programming issues, such as integrating within existing applications, focusing on modelling and analysing results and so is therefore ideal for prototyping.

This modelling environment is implemented in C++ and is to be released as an open source standalone library.

References

[1] A. Griewank, ‘‘On Automatic Differentiation,’’ pp. 83–107 in *Mathematical Programming*, ed. M. Iri and K. Tanabe, Kluwer Academic Publishers (1989).

[2] Automatic Differentiation of Algorithms: Theory, Implementation and Application, ed. A. Griewank and G. Corliss, SIAM (1991).

[3] Masao Iri, ‘‘History of Automatic Differentiation and Rounding Error Estimation’’ pp. 3–16 in *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, ed. A. Griewank and G. F. Corliss, SIAM (1991).

[4] David M. Gay, ‘‘Automatic Differentiation of Nonlinear AMPL Models’’ pp. 61–73 in *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, ed. A. Griewank and G. F. Corliss, SIAM (1991).

[5] B. Christianson, ‘‘Automatic Hessians by Reverse Accumulation’’ *IMA J. Numer. Anal.* 12 (1992), pp. 135–150.

[6] Riccardo Spagna “Notes on the calculation of the derivatives for least-squares crystal structure refinement”, International Union of Crystallography, Newsletter No.7, November 2006.

[7] Giacovazzo, C. (2002), *Fundamentals of Crystallography*, 2nd edn. Oxford: Oxford University Press.

[8] Sheldrick, G.M. (1997), *SHELXL97. “Program for the Refinement of Crystal Structures”,* University of Goettingen, Germany.

Acknowlegements

I am grateful to David Watkin for his support and help in preparation of this article and I thank Amber Thompson for providing all necessary data and testing, both from *Chemical Crystallography Laboratory, University Of Oxford.*

I gratefully acknowledge the financial support of EPSRC (Grant number: EP/C536274/1) for the Age Concern project..

OLEX-2

Oleg V. Dolomanov, Luc J. Bourhis, Richard J. Gildea, Judith A. K. Howard and Horst Puschmann

Department of Chemistry, University Science Laboratories, South Road, Durham, DH1 3LE, United Kingdom. E-mail: horst.puschmann@durham.ac.uk

Abstract

Olex2 is a modern graphical user interface (GUI) that provides all the tools necessary to perform small molecule crystal structure analysis and visualisation. It is fully compatible with all of the SHELX[1] family of crystallographic solution and refinement programs and provides a user interface to the solution and refinement capabilities of the cctbx[2] as developed by this same Durham group and presented in a separate article in this newsletter.

Introduction

The underlying concepts and principles of Olex2 have been reported in a paper published in J. Appl. Cryst in April 2009[3]. In this short news article presented here, we will not repeat what has been said previously; we have two other goals in mind: firstly, we wish to draw attention to the existence of this freely available and open-source (BSD license) software and encourage the reader to download, install and evaluate this platform-independent package. Secondly, we wish to put Olex2 in context with the other aspects of the EPSRC research grant[4] that funds this project.

Historic Background

Much of the software landscape surrounding the area of small molecule crystallography has become virtually stagnant. At the turn of the century, the situation looked particularly bleak: Apart from the SHELXL/XH, the only other refinement software with any significant distribution among practising structural analysts was CRYSTALS[5]. For structure solution, the use of SHELXS is much more widespread than its only competitor: SIR[6]. As has been pointed out by David Watkin in a lecture at the ECM 2007 in Marrakech[7], the situation has not always been like this. There was a time spanning the 1960s and 1970s, where innovation in crystallographic programming was ripe and software for structure analysis was being developed in almost every laboratory. Some of this software subsequently found a wider distribution. During the 1980s, the crystallographic community had at their disposal a wide variety of well-suported and fully functional crystallographic systems like X-ray[8], CRYSTALS, XTL[9], SHELX, SDP[10], RONTGEN 75[11], DIRDIF[12], XTAL[13], CRYSTAN[14], NRCVAX[15] and many more. So, what happened during the 1990s, that led to the demise of all but a few of these? Why is it that the programs that have survived, tend to be those that were written mainly by one author and with a relatively narrow task in mind? And why has XTAL, a crystallographic system encompassing some 50 programs, 30-odd programmers and a very competent management structure, not passed the test of time? The answer is obvious: without the internet and all the radical new features that inevitably followed, meaningful collaborative work on a big software project was almost impossible then. Today, we have instant communication through e-mail and messenger services, sophisticated code versioning systems, access to programming documentation as well as access to all relevant scientific papers - and all in real time. This, much more so than advances in computer and programming technologies (much as these play an important role), is the real difference between then and now. To conclude from the sudden demise of virtually all collaborative crystallographic computing efforts from the 1960s and 1970s that collaborations do not work is simply a fallacy. In fact, virtually all today's successful projects in the area of protein crystallography are the result of big collaborative efforts for example CCP4[16], CNS[17] and PHENIX[18].

Project Philosophy

Olex2 is a collaborative project. Our aim is to make Olex2 a one-stop program from which all crystallographic tasks can be managed easily. This is simply too large task to be performed by one person alone. All authors of this article fulfil an active role in the generation and maintenance of the Olex2 code. We are working closely with the cctbx team on many aspects of the project. Because of the open-source nature of the project, we are also increasingly seeing contributions from interested members of the crystallographic community.

The reason for including a Graphical User Interface (GUI) project such as Olex2 in the original proposal that led to the funding of the 'Age Concern: Crystallographic Computing for the Future' project, is that we felt the need to present the results that we have obtained from our work with the smtbx (see our other 'Durham' contribution in this newsletter) as well as the work of the Oxford team to a wider crystallographic audience in a practical and meaningful way.

We see Olex2 as a GUI from which - eventually - a large variety of programs can be addressed in a unified way. The internal structure model format is all-encompassing, so that Olex2 can generate an input file in a variety of formats, run an external program and then read the program output back into the Olex2 structure model format without the loss of information. Of course, for more tightly integrated routines, namely those originating from this 'Crystallographic Software for the Future' project, there is no need to communicate via a file in/out system.

Design Principles

Program Design

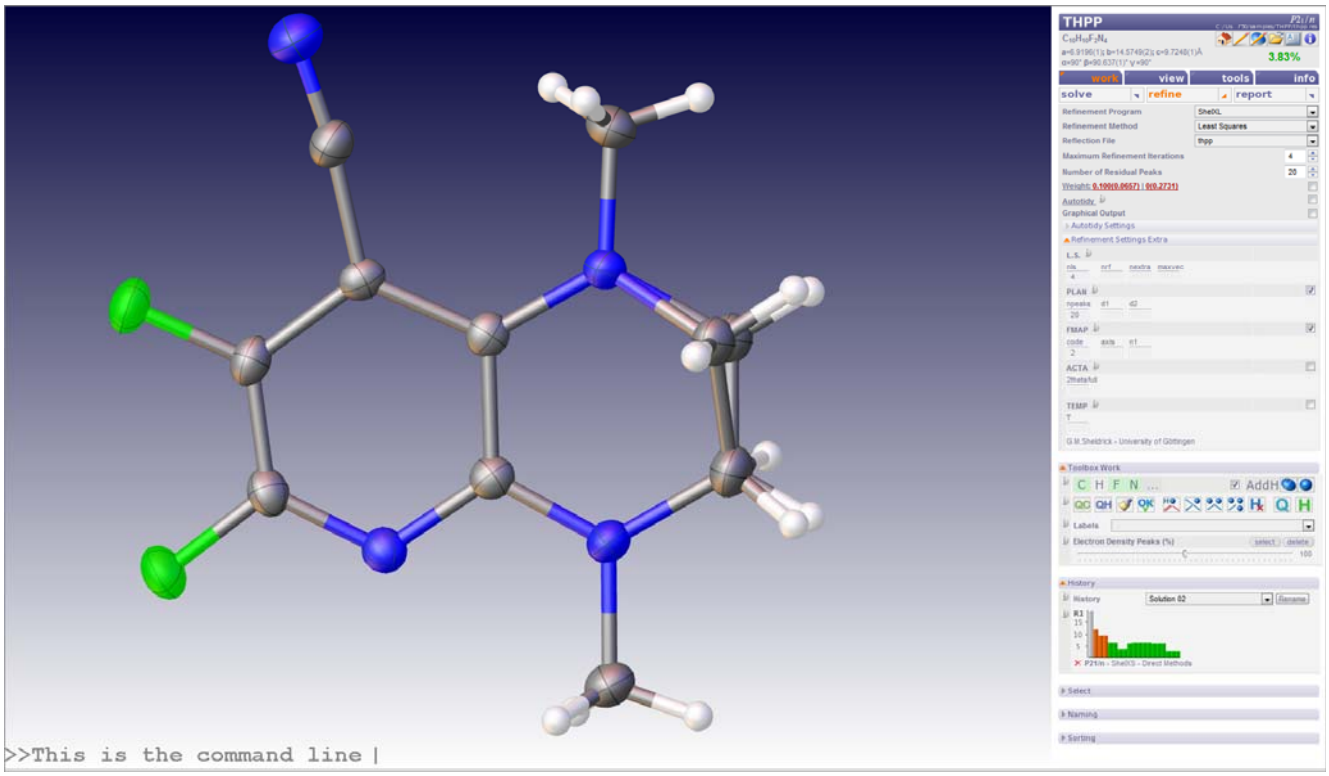
Olex2 is designed for maximum flexibility. The core of the program is written in the C++ programming language and is highly optimised. The core deals with the internal structure and refinement model, symmetry and the display of the structure (via OpenGL[19]). The functionality of the Olex2 core can be extended either using a built-in macro language or by utilising the scripting Python language[20], which allows interaction with the core on a highly sophisticated level. The GUI itself is written in Python and the GUI display is generated using wxWidgets[21] and extended HTML.

GUI Design

All aspects of the GUI have been designed with the user in mind. The focus is made on what the user wants to achieve. This represents a major deviation from the GUI design paradigm of most existing crystallographic software. We focus on a clear and simple layout of the GUI and the goal is to enable access to even the most powerful features with a minimum of technical knowledge. There are exactly two windows in Olex2 - the *main window*, which contains the OpenGL representation of the current structure model and the command line, and then there is the *control panel*, which contains the user tools. These tools are available in structure solution, refinement, report generation and structure information tabs, and are designed to harvest all the necessary information before a process is started, so there is no need for pop-up boxes in Olex2 at all.

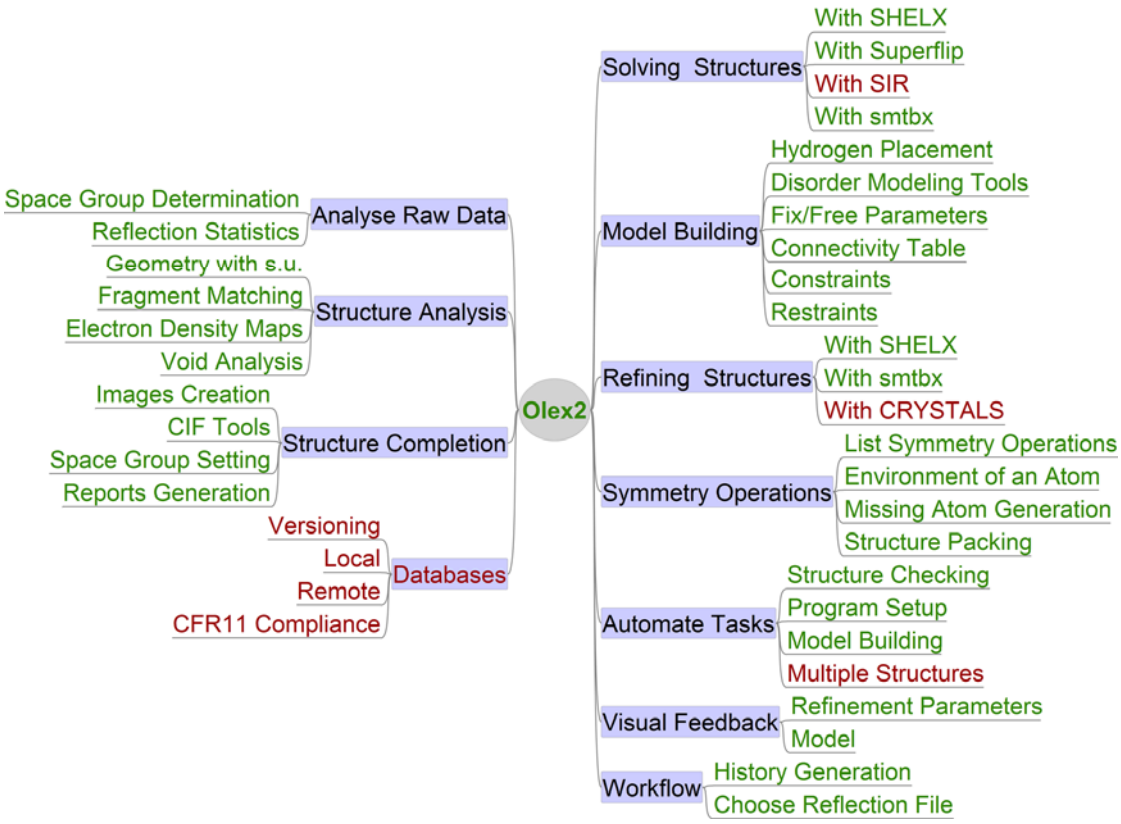
The structure model is manipulated in a standard way, regardless of the the program performing the structure solution or refinement. At the same time, it is possible to access all specific settings for any program.

This design approach of Olex2 suits crystallographers of any ability from the new through to expert user. It has also been shown in numerous workshops around the world that Olex2 is more readily accepted by students than other crystallographic software primarily because the learning curve is very gentle and students can see the result with a click of the button, be it structure solution or refinement.



Features of Olex2

An overview of the functions contained in Olex2 is given below. The graph is constantly expanding as new features are added. The graph was generated using FreeMind[22], and the source file for this graph is included with the Olex2 distribution. Using FreeMind, all items in the graph below will 'expand' to give more detailed information. We have attempted to group the many tools that are currently available in Olex2 into logical units, which are represented below. Items marked as red are on our roadmap, but have not yet been implemented.



Availability and Support

Olex2 is an open source project and all source code is available under the BSD licence from our Sourceforge (Olex2 core) and Durham (Olex2 GUI) repositories. We maintain a portal at www.olex2.org, where compiled versions of the software can be downloaded for Windows, Mac OSX and common Linux installations. Olex2 supports multiple languages, including Chinese.

We are aiming at making Olex2 fully self-documenting: all items on the GUI will soon have a link to a help file that can be displayed in-line. A detailed document encompassing all Olex2 commands together with their options is currently in preparation. Snapshots of this document are shipped with the Olex2 installation. There are also extensive resources on www.olex2.org and also in our Olex2 forum hosted on the xForum[23].

References

1 Sheldrick, G. M. (2008). *Acta Crystallogr.*, **A64**, 112–122.
2 http://cctbx.sourceforge.net/
3 Dolomanov, O. V., Bourhis, L. J., Gildea, R. J., Howard, J. A. K. and Puschmann, H. (2009). *J. Appl. Cryst.*, **42**, 339–341.
4 EPSRC grant number EP/C536274/1.
5 Betteridge, P. W., Carruthers, J. R., Cooper, R. I., Prout, K., Watkin, D. J. (2003). *J. Appl. Cryst.*, **36**, 1487.
6 Giacovazzo, C, et al. (1989) *J. Appl. Cryst.,* **22**, 389-393.
7 Watkin, D.J., ECM24 Marrakech, Small Molecule Computing.
8 X-ray: Stewart J. M., Kundell, F. A. & Baldwin, J. C., 1963.
9 XTL: Sparks, 1972.
10 SDP: Okaya, Y. & Frenz, B. A., 1975.
11 RONTGEN: Andrianov, V. I., Safina, Z. Sh. & Tarnopol'skii, B. L., 1975.
12 DIRDIF: Gould, R. O., van den Hark, Th. E. M. & Beurskens, G., 1975.
13 XTAL: Hall, S. R. & Stewart, J. M., 1978.
14 CRYSTAN: Burzlaff, Bohem & Gomm, 1978.
15 NRCVAX: Gabe, E. J., Lee, F. L. & Le Page, Y., 1987.
16 CCP4: http://www.ccp4.ac.uk/.
17 CNS: http://cns-online.org/v1.21/.
18 PHENIX: http://www.phenix-online.org/.
19 Khronos Group (2008). Open-GL – the Industry Standard for High Performace Graphics, http://www.opengl.org/
20 Python Software Foundation (2008). Python Programming Language, http://www.python.org/
21 wxWidgets (2008). wxWidgets – Cross-Platform GUI Library, http://www.wxwidgets.org/
22 FreeMind, Joerg Mueller, http://freemind.sourceforge.net/
23 Warren, J. E., http://www.x-rayman.co.uk.

CRYRM

David Duchamp[1], Larry Henling[2] & Richard Marsh[2]

1) 6209 Litchfield Lane Kalamazoo, Michigan 49009, USA. E-mail: djduchamp@mac.com ; 2) Beckman Institute, 139-74 California Institute of Technology Pasadena, California 91125, USA. E-mail: lmh@caltech.edu & rmarsh@caltech.edu

The CRYRM system of crystallographic programs was designed and eveloped in the early 1960's in the laboratory of Dr. Richard Marsh at Caltech. A team of postdocs and graduate students, including David Duchamp, Ned Webb, Carlo Grammaccoli, and George Reeke, worked under Dick's guidance to create this unique system. CRYRM used a system approach to crystallographic computing, where all component routines were written to use common files for input data and results, making it very easy to proceed from one calculation to another in the same computer run. Crystallographic programs for carrying out various calculations were available at the time; for the most part, however, these were stand-alone programs with input data needing to be re-formatted and re-punched before each calculation. The primary purpose for developing CRYRM was to have an integrated set of programs where the various steps in structure determination--data collection and reduction, Patterson and Fourier maps, least-squares refinement, and calculations of molecular geometry--would all operate under a master program with a common format for the input and output data. This, coupled with crystallographer-friendly input commands, allowed crystallographers to focus on their science instead of constantly fighting the computer.

A major hurdle in developing such a process was the problem of interfacing between the master program and the individual routines, since computer memory was very limited and the entire program could hardly survive in the presence of, say, a least-squares matrix. The target computer was the IBM 7094, then a calculating powerhouse but a wimp in memory and speed by today's standards. To make maximum use of the IBM 7094's memory and computing power, CRYRM was written in 7094 assembly language. The various computing routines were brought into computer memory only when needed. Portions of the IBM/Caltech operating system that were not needed by CRYRM were rolled out to disk to make room for least squares matrices and Fourier maps, then automatically rolled back when calculations were complete.

CRYRM included many innovations, a number of which are now incorporated into current crystallographic systems. Multiple observations of intensity data--originally provided by visual estimates from multiple-film Weissenberg photographs, and later from output from an automated diffractometer--were corrected for Lorentz and polarization effects, sorted and averaged, and weighted appropriately using agreement statistics from multiple observations of the same intensity. This process is similar to those used today to process area detector data. Fourier and Patterson sections could be calculated in any general plane using an efficient scheme for calculating the actual points, rather than interpolating between points in a three-dimensional map. Space-group-specific expressions were used for structure factor and Fourier calculations, to speed up calculations; the crystallographer needed only to specify the space group number from the International Tables. The first printed manual describing how to use the CRYRM system was issued in 1964.

In least squares, CRYRM provided for "multiple matrix" calculations for those cases where there was not enough computer memory to hold the entire refinement matrix. Prior to CRYRM's multiple matrix capability, least squares was either block diagonal (each atom's parameters in a separate matrix) or full matrix. Multiple matrices allowed the crystallographer to take into account anticipated correlations between designated atoms when full-matrix calculations were not possible. CRYRM least-squares calculations were based on values of F(obs) squared rather than F(obs), an uncommon procedure at the time since it complicated the calculations somewhat but one that permitted realistic weighting of the observations. Error propagation was coded into all molecular geometry calculations to give accurate standard deviations in derived descriptors, based on standard errors and correlations among the refined parameters.

In subsequent years, CRYRM was rewritten in Fortran, mainly by David Duchamp, renamed CRYM to differentiate it from the original, and made to run on many different computer systems. All features, except for the processing of intensities from film data, were retained. New features were added, such as absorption corrections, peak picking in electron density and Patterson maps, anisotropic Wilson calculations, and "RCHECK" for examining the F(obs)-F(cal) agreement as a function of index-parity groups. When Isabella Karle began to demonstrate success solving structures using direct methods, a direct-methods routine, "DIREC", was added to CRYM. DIREC used the symbolic addition method with triplets and quartets developed by Jerry Karle and Herb Hauptman, and was periodically updated to make use of Herb's latest work.

Over the years, many crystal structures were processed from diffractometer data through final refinement using CRYM, often in one computer run. Today CRYM has largely been replaced by newer crystallographic computing systems. Thanks to the efforts of Larry Henling, CRYM is still running at Caltech, where it is occasionally used for molecular geometry calculations.

Many people have participated in the care and feeding of CRYM: care in keeping the programs operating during decades of major changes in computer facilities while feeding the system with new or more convenient programs. Besides those already cited, prominent names during this process include Ivars Ambats, Bill Connick, Howard Einspahr, Kathy Flanagan, Bill Schaefer, Verner Schomaker, Kirby Slagle, Dick Stanford, and Jean Westphal. They and others deserve the thanks of generations of CRY(R)M users..

Editor’s note: CRYRM manuals are listed starting on page 2 and page 197 within Addendum B: Computing Software manuals and reference materials

DIMS - Direct methods In Multidimensional Space

Hai-fu Fan & Yuan-xin Gu
Institute of Physics, Chinese Academy of Sciences, Beijing 100190, P.R. China. E-mail: fanhf@cryst.iphy.ac.cn *; WWW:* http://cryst.iphy.ac.cn/ *;* http://cryst.iphy.ac.cn/VEC/download.php

1. Introduction

DIMS (Direct methods In Multidimensional Space) (Fu & Fan, 1994; Fan, 2005a,b) is a direct-methods program for solving one-dimensionally modulated incommensurate structures and composite structures consists of two subsystems with two axes of the unit cell coincided to each other. Actually DIMS can also deal with three-dimensional periodic structures including commensurate modulated structures. The latter is also known as superstructures. The theoretical basic behind DIMS is the direct-method treatment of the phase ambiguity due to pseudo-translational symmetry. This originated from solving the crystal structure of a natural amino acid [Fan, 1975 (in Chinese); see also Fan, 1984]. The crystal structure in that study possesses a two-fold pseudo-translational symmetry leading to phase ambiguities for one half of the total reflections. The Sayre equation (Sayre, 1952) was modified and successfully used to break the phase ambiguity. Later, a direct method of solving commensurate modulated structures (superstructures) was proposed [Fan, He, Qian and Liu, 1978 (in Chinese); see also Fan, Yao, Main & Woolfson, 1983]. The method was subsequently extended to multidimensional space for dealing with incommensurate modulated structures (Hao, Liu & Fan, 1987). Further developments were made on extending the method in solving composite structures (Fan, Smaalen, Lam & Beurskens, 1993; Sha *et al.*, 1994; Mo *et al.*, 1996). DIMS has been integrated with the program VEC (Visual computing in Electron Crystallography) (Wan *et al.* 2003). Implementation and application of DIMS/VEC have been described in detail previously (Fan, 2005a, b). In this paper a summary of the theory behind DIMS and some details inside DIMS will be given.

2. Theory behind DIMS

2.1 Modulated structures

Modulated structures can be regarded as the result of applying a periodic modulation to a basic structure, which possesses exact 3-dimensional periodicity. Figure 1 shows two examples. The modulation wave in Fig.1a represents the fluctuation of atomic occupancy. When it is applied to a basic structure (atoms of which are represented by black vertical rods), the 'heights' of the atoms are modified. A commensurate modulated structure (superstructure) will result (Fig.1b), if the period **T** of the modulation function matches the period **t** of the basic structure, i.e. **T/t** = n, where n is a simple integer. The resulting superstructure now has a true period **T** and a pseudo period **t**, which respectively correspond to a true unit cell and a pseudo unit cell. On the other hand, if **T** does not match **t** (Fig.1c), i.e. **T/t** = r, where r is not a simple integer, we obtain an incommensurate modulated structure, which possesses no exact 3-dimensional periodicity although **t** remains a pseudo period. A modulation function can also represent the fluctuation of atomic position or thermal motion. In practice a modulated structure may simultaneously include different kinds of modulation.

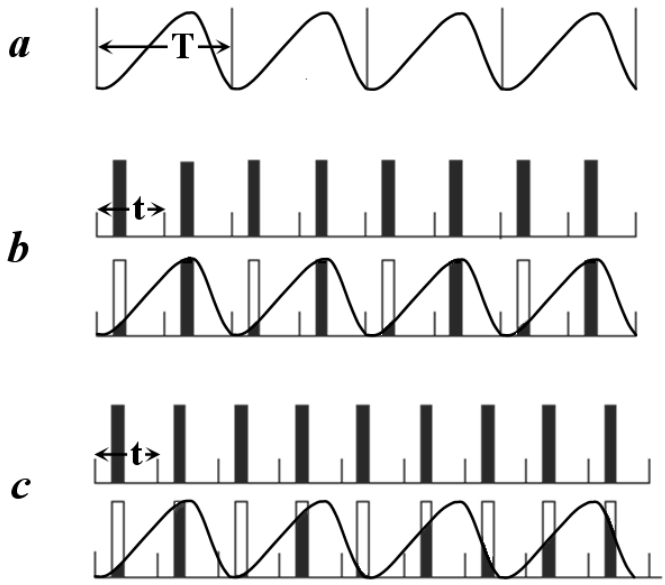


Figure 1. Occupational modulation of a one-dimensional structure

Incommensurate modulated crystals yield 3-dimensional diffraction patterns, which contains satellites round the main reflections. An example of a section of such a 3-dimensional diffraction pattern is shown schematically in Fig.2. The main reflections are consistent with a regular 3-dimensional reciprocal lattice but the satellites do not fit the same lattice. On the other hand, while the satellites do not match the main lattice, they have their own periodicity (see the vertical line segments in Fig.2). Hence, it can be imagined that the 3-dimensional diffraction pattern is the projection of a 4-dimensional reciprocal lattice, in which the main and the satellite reflections are all regularly situated at the lattice nodes. From the properties of the Fourier transform the incommensurate modulated structure here considered can be regarded as a 3-dimensional "section" of a 4-dimensional periodic structure. The above example corresponds to a one-dimensional modulation. In the case of n-dimensional (n=1, 2, …) modulation, it needs a (3+n)-dimensional description.

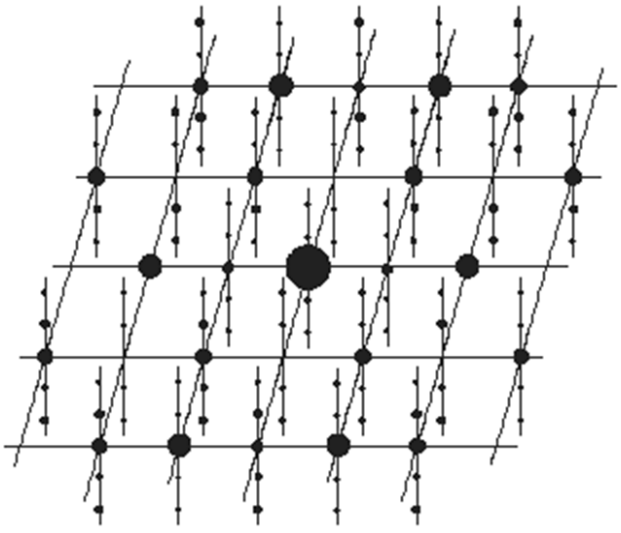


Figure 2. Schematic diffraction pattern of an incommensurate modulated structure

There is a special kind of incommensurate modulated structures called composite structures. The characteristic of which is the coexistence of two or more mutually incommensurate 3-dimensional lattices. Owing to the interaction of coexisting lattices, composite structures are also incommensurate modulated structures. Unlike ordinary incommensurate modulated structures, composite structures do not have a 3-dimensional basic structure. Instead, they will have a 4- or higher-dimensional basic structure.

For details of superspace representation of modulated crystal structures the reader is referred to Janssen *et al*. (2006), Smaalen (1995) and Yamamoto (1996).

2.2 4-dimensional representation of one-dimensionally modulated structures

As is described above, one-dimensionally incommensurate modulated structures can be regarded as a 4-dimensional periodic structure cut with a 3-dimensional hyperplane parallel to the 3-dimensional physical space. A position vector **x** within the 4-dimensional unit cell is expressed as

$$\mathbf{x} = x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + x_3\mathbf{a}_3 + x_4\mathbf{a}_4 . \qquad (1)$$

In equation (1), x_1, x_2, x_3 and x_4 are fractional coordinates; $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$ and $\mathbf{a}_4$ are basic vectors defining the 4-dimensional unit cell. A reciprocal lattice vector **h** is expressed as

$$\mathbf{h} = h_1\mathbf{b}_1 + h_2\mathbf{b}_2 + h_3\mathbf{b}_3 + h_4\mathbf{b}_4 . \qquad (2)$$

In equation (2), h_1, h_2, h_3 and h_4 are diffraction indices, which are components of **h** with respect to the basic vectors $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ and $\mathbf{b}_4$ of the 4-dimensional reciprocal lattice. Vectors $\mathbf{a}_i$ and $\mathbf{b}_j$ satisfy the reciprocal relationships

$$\mathbf{a}_i \cdot \mathbf{b}_j = \delta_{ij} \quad (i, j = 1,2,3,4), \ \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} . \qquad (3)$$

The 4-dimensional unit cell of the modulated structure is related to the 3-dimensional unit cell of the basic structure through the following relationships.

$$\begin{aligned} \mathbf{a}_1 &= \mathbf{a} - q_1\mathbf{d} & \mathbf{b}_1 &= (\mathbf{a}^*, 0) \\ \mathbf{a}_2 &= \mathbf{b} - q_2\mathbf{d}, & \mathbf{b}_2 &= (\mathbf{b}^*, 0). \\ \mathbf{a}_3 &= \mathbf{c} - q_3\mathbf{d} & \mathbf{b}_3 &= (\mathbf{c}^*, 0) \\ \mathbf{a}_4 &= (0, \mathbf{d}) & \mathbf{b}_4 &= (\mathbf{q}, \mathbf{d}) \end{aligned} \qquad (4)$$

Where **a**, **b**, and **c** are the basic vectors defining the unit cell of the 3-dimensional basic structure, while **a***, **b***, and **c*** are the basic vectors defining the corresponding reciprocal lattice. The unit vector **d** is perpendicular to the 3-dimensional physical space. The modulation wave vector **q** is expressed as

$$\mathbf{q} = q_1\mathbf{a}^* + q_2\mathbf{b}^* + q_3\mathbf{c}^* . \qquad (5)$$

In the case of incommensurate modulation, at least one of the components q_1, q_2 and q_3 on the right-hand side of (5) should be irrational.

The structure-factor formula for a one-dimensionally modulated structure is written as

$$F(\mathbf{h}) = \sum_{j=1}^{N} f_j(\mathbf{h}) \exp\left[i2\pi \left(h_1\bar{x}_{j1} + h_2\bar{x}_{j2} + h_3\bar{x}_{j3} \right) \right] . \qquad (6)$$

Where

$$f_j(\mathbf{h}) = f_j^o(h)\int_0^1 P_j(\bar{x}_4)\exp\left[i2\pi\left(h_1U_{j1} + h_2U_{j2} + h_3U_{j3} + h_4x_{j4}\right)\right]d\bar{x}_4 \cdot \qquad (7)$$

On the right-hand side of (7) $f_j^o(h)$ is the ordinary atomic scattering factor, P_j is the compositional modulation function and U_j describes the deviation of the j^{th} atom from its average position ($\bar{x}_{j1}$, $\bar{x}_{j2}$, $\bar{x}_{j3}$). For more details on (6) and (7) the reader is referred to the papers by de Wolff (1974), Yamamoto (1982) and Hao, Liu & Fan (1987). What should be emphasized here is that, according to (6) and (7) a modulated structure can be regarded as a set of 'modulated atoms' situated at their average positions in 3-dimensional space. While the 'modulated atom' in turn is defined by the 'modulated atomic scattering factor' $f_j(\mathbf{h})$.

2.3 Sayre's equation in multidimensional space

It has been proved by Hao, Liu & Fan (1987) that the Sayre equation (Sayre, 1952) can easily be extended into multidimensional space. We have

$$F(\mathbf{h}) \approx \frac{\theta}{V}\sum_{\mathbf{h'}} F(\mathbf{h'})F(\mathbf{h}-\mathbf{h'}) \cdot \qquad (8)$$

Where θ is an atomic form factor; V is the unit-cell volume of the basic structure. The reciprocal-lattice vector **h** is now a multidimensional vector defined as

$$\mathbf{h} = \sum_{i=1}^{3+n} h_i\mathbf{b}_i \qquad , \qquad n = 1, 2, 3, \cdots \cdot \qquad (9)$$

Where h_i are components of the vector **h**; $\mathbf{b}_i$ are basic vectors defining a multidimensional reciprocal unit cell. The right-hand side of (8) can be split into three parts, i.e.

$$F(\mathbf{h}) = \frac{\theta}{V}\left\{\sum_{\mathbf{h'}} F_m(\mathbf{h'})F_m(\mathbf{h}-\mathbf{h'}) + 2\sum_{\mathbf{h'}} F_m(\mathbf{h'})F_s(\mathbf{h}-\mathbf{h'}) + \sum_{\mathbf{h'}} F_s(\mathbf{h'})F_s(\mathbf{h}-\mathbf{h'})\right\} \cdot \qquad (10)$$

In equation (10), the subscript *m* stands for main reflections, while the subscript *s* stands for satellites. Since the intensities of satellites are on average much weaker than those of main reflections, the last summation on the right-hand side of (10) is negligible in comparison with the second, while the last two summations on the right-hand side of (10) are negligible in comparison with the first. If F(**h**) on the left-hand side of (10) represents structure factors of main reflections, we have to first approximation

$$F_m(\mathbf{h}) \approx \frac{\theta}{V}\sum_{\mathbf{h'}} F_m(\mathbf{h'})F_m(\mathbf{h}-\mathbf{h'}) \cdot \qquad (11)$$

This implies that the basic structure can be solved by conventional methods (direct methods or other methods) in 3-dimensional space neglecting all satellite reflections. On the other hand, if F(**h**) on the left-hand side of (10) corresponds to satellite reflections, it follows that

$$F_s(\mathbf{h}) \approx \frac{\theta}{V}\sum_{\mathbf{h'}} F_m(\mathbf{h'})F_m(\mathbf{h}-\mathbf{h'}) + \frac{2\theta}{V}\sum_{\mathbf{h'}} F_m(\mathbf{h'})F_s(\mathbf{h}-\mathbf{h'}) \cdot \qquad (12)$$

For ordinary incommensurate modulated structures the first summation on the right-hand side of (12) has vanished. Because any three-dimensional reciprocal lattice vector corresponding to a main reflection will

have zero components in the extra dimensions so that the sum of two such lattice vectors could never give rise to a lattice vector corresponding to a satellite. We then have

$$F_s(\mathbf{h}) \approx \frac{2\theta}{V}\sum_{\mathbf{h'}} F_m(\mathbf{h'})F_s(\mathbf{h}-\mathbf{h'})\cdot \qquad (13)$$

For composite structures (Fan *et al.* 1993; Sha *et al.*, 1994; Mo *et al.*, 1996) on the other hand, since the average structure itself is a 4- or higher-dimensional periodic structure, the first summation on the right-hand side of (12) does not vanish. We have instead of (13) the following equation:

$$F_{\mathrm{s}}(\mathbf{h}) \approx \frac{\theta}{V}\sum_{\mathbf{h'}} F_{\mathrm{m}}(\mathbf{h'})F_{\mathrm{m}}(\mathbf{h}-\mathbf{h'})\cdot \qquad (14)$$

Equation (13) or (14) imply that, once the phases of main reflections are known, it is straight forward to derive phases of satellites by using (13) and (14) respectively for ordinary incommensurate modulated structures and composite structures. The whole procedure will be in the following stages:

i) Derive the phases of main reflections using Equation (11) or other methods in 3-dimensional space;
ii) Derive the phases of satellite reflections using Equation (13) or (14);
iii) Calculate a multidimensional Fourier map using the observed structure-factor magnitudes and the phases from i) and ii);
iv) Cut the resulting Fourier map with a 3-dimensional 'hyperplane' to obtain a 'structure image' of the incommensurate modulated structure in the 3-dimensional physical space;
v) Parameters of the modulation functions are measured directly on the multidimensional Fourier map resulting from iii).

3. (3+1)-dimensional symmetry generators from superspace-group symbols

In the first edition of DIMS (Fu & Fan, 1994; source code in Appendix I) the superspace symmetry generators should be manually input to the program. Later, the subroutine SPGR4D (Fu & Fan, 1997; source code in Appendix II) was written and incorporated into DIMS enabling automatic derivation of superspace symmetry generators from the input two-line superspace-group symbol (Wolff, Janssen & Janner, 1981). Recently, the subroutine SYMBOL1to2 was written for converting one-line symbols to two-line symbols of (3+1)-dimensional superspace groups (Li, Li & Fan, 2009; source codes in Appendix III). The subroutine has been incorporated into the program VEC (Wan *et al.*, 2003). Now, the program DIMS (source code of MS Windows version in Appendix IV) invoking from VEC, accepts symmetry generators, two-line superspace-group symbols or one-line superspace-group symbols.

4. Phasing formula

The phasing formula used in DIMS looks like the tangent formula of Karle & Hauptman (1956)

$$\tan\varphi(\mathbf{h}) = \frac{\sum_{\mathbf{h'}}\left|E(\mathbf{h'})E(\mathbf{h}-\mathbf{h'})\right|\sin\left[\varphi(\mathbf{h'})+\varphi(\mathbf{h}-\mathbf{h'})\right]}{\sum_{\mathbf{h'}}\left|E(\mathbf{h'})E(\mathbf{h}-\mathbf{h'})\right|\cos\left[\varphi(\mathbf{h'})+\varphi(\mathbf{h}-\mathbf{h'})\right]}, \qquad (15)$$

with

$$\left|E(\mathbf{h})\right| \propto \left|F(\mathbf{h})\right| \Big/ \left\langle \left|F(\mathbf{h})\right|^2 \right\rangle^{1/2}. \qquad (16)$$

Here the vector **h** is a 4-dimensional reciprocal lattice vector. Strictly speaking, equation (15) is not a tangent formula. The tangent formula for 3-dimensional periodic structures may be regarded as the result of maximizing the phase probability density function given by Cochran (1955):

$$P(\varphi_{\mathbf{h}}) = \left[2\pi I_0(\alpha)\right]^{-1} \exp[\alpha \cos(\varphi_{\mathbf{h}} - \langle\varphi_{\mathbf{h}}\rangle)] \qquad (17)$$

with

$$\alpha = \left[\left(\sum_{\mathbf{h}} \kappa_{\mathbf{h},\mathbf{h}'} \sin(\varphi_{\mathbf{h}} + \varphi_{\mathbf{h}-\mathbf{h}})\right)^2 + \left(\sum_{\mathbf{h}} \kappa_{\mathbf{h},\mathbf{h}'} \cos(\varphi_{\mathbf{h}} + \varphi_{\mathbf{h}-\mathbf{h}})\right)^2\right]^{1/2} \qquad (18)$$

and

$$\kappa_{\mathbf{h},\mathbf{h}'} = 2\sigma_3\sigma_2^{-3/2}\left|E_{\mathbf{h}}E_{\mathbf{h}'}E_{\mathbf{h}-\mathbf{h}'}\right|. \qquad (19)$$

However this phase probability density function is not necessarily valid for incommensurate modulated structures. Besides, in practice all observable reflections [with the restriction of equation (11), (13) or (14)] are used in the calculation of equation (15). In view of these, equation (15) would better be regarded as the angular portion of the Sayre equation rather than the tangent formula (see the discussion by Fan, 1998). On the other hand, as is shown by Gelder *et al.* (1996) that the probability distribution associated with the structure invariants $E(-\mathbf{h})E(\mathbf{h}')E(\mathbf{h}-\mathbf{h}')$, where **h** and **h'** are reciprocal vectors of main and/or satellite reflections, approximately has the same functional form as the Cochran distribution. Hence, equation (15) can still be approximately regarded as the tangent formula and, the associated equations (16) to (19) can still be used in dealing with incommensurate modulated structures. The $|E(\mathbf{h})|$ values calculated from equation (16) are not exactly the normalized structure-factor amplitudes but just specially scaled structure-factor amplitudes, which contain an empirical scaling factor to balance between main and satellite reflections. The use of equation (16) implies also that the calculation is independent of atomic scattering factors. Hence for ordinary incommensurate modulated structures DIMS can be used in phasing X-ray, electron and even neutron diffraction data. Equation (18) is used empirically for the calculation of figures of merit. However it should be aware that such figures of merit are not as reliable as that used for 3-dimensional periodic structures.

5. Phasing strategy

The phasing strategy used in DIMS is the 'random-starting-phases multisolution algorithm' developed in M.M. Woolfson's group in York University, UK. The reader is referred to the paper by Yao (1981) for details. For ordinary incommensurate modulated structures, it is assumed that phases of main reflections are already known before running DIMS. These phases are treated as 'known phases' and are kept fixed during the phasing process. Phases of all satellite reflections are 'unknown phases', which are given random starting values and will be refined during the phasing process. For composite structures, phases of main reflections can either be input as known phases or derived by DIMS it self before phasing satellite reflections. All unknown phases are given random starting values and then refined using the phasing formula.

5.1 Phasing satellites of ordinary incommensurate modulated structures

Here, the most important part is to phase the first-order satellites, since higher-order satellites are usually much weaker. The phasing is based on phase relationships existing in equation (13). Since phases of the main reflections are known in advance, the phasing process is straightforward. After deriving phases of the first-order satellites, phases of higher-order satellites (if any) will be derived also based on phase relationships existing in equation (13). In addition, phase relationships involving three satellites with at least one first order satellite will also be used. The goal of using this kind of phase relationships is to link phases between the first-order and higher order satellites.

5.2 Phasing main reflections of composite structures

The phasing is based on phase relationships existing in equation (11). The process is nearly the same as that for 3-dimensional periodic structures. Here the normalized structure-factor amplitudes are calculated as

$$|E(\mathbf{h})| = |F(\mathbf{h})| \Big/ \left(\varepsilon \sum_{j=1}^{N} f_j^2 \right)^{1/2} \qquad (20)$$

with atomic scattering factors for X-rays. In this case DIMS can deal with only X-ray diffraction data. For details of the procedure, the reader is referred to the paper by Mo *et al.* (1996).

5.3 Phasing satellite reflections of composite structures

The phasing is based on phase relationships existing in equation (14). Phases of the main reflections are known and kept fixed during the phasing process. For details of the procedure, the reader is referred to the paper by Fan et al. (1993).

Acknowledgement

The project was supported in part by the National Natural Science Foundation of China.

Software Download

http://cryst.iphy.ac.cn/ ; http://cryst.iphy.ac.cn/VEC/download.php

References

Cochran, W. (1955). Relations between the Phases of Structure Factors. *Acta Cryst.* **8**, 473-478.

Fan, H.F. (1975). The phase ambiguity in heavy-atom method treated by structure-factor relationships. *Acta Phys. Sin.* **24**, 57-60. (In Chinese)

Fan, H.F. (1984). The problem of phase ambiguity in single crystal structure analysis. *The Rigaku Journal*, **1**, *No.* 2, 15-21.

Fan, H.F. (1998). Sayre equation, tangent formula and SAYTAN. in *Direct Methods for Solving Macromolecular Structures*. Ed. by S. Fortier, Kluwer Academic Publishes, The Netherlands. Pp. 79-85.

Fan, H.F. (2005a). DIMS on the VEC platform. *IUCr Computing Commission Newsletter* No.5, 16-23.

Fan, H.F. (2005b). DIMS/VEC applications. *IUCr Computing Commission Newsletter* No.5, 24-27.

Fan, H.F., He, L., Qian, J.Z. & Liu, S.X. (1978). Solving crystal superstructures by the direct method. *Acta Phys Sin.* **27**, 554-558. (In Chinese)

Fan, H.F., van Smaalen, S., Lam, E.J.W. & Beurskens, P.T. (1993). Direct methods for incommensurate intergrowth compounds I. Determination of the modulation. *Acta Cryst*. A**49**, 704-708.

Fan, H.F., Yao, J.X., Main, P. & Woolfson, M.M. (1983). On the application of phase relationships to complex structures. XXIII. Automatic determination of crystal structures having pseudo-translational symmetry by a modified MULTAN procedure. *Acta Cryst.* A**39**, 566-569.

Fu, Z.Q. & Fan, H.F. (1994). DIMS --- a direct method program for incommensurate modulated structures. *J. Appl. Cryst.,* **27**, 124-127.

Fu, Z.Q. & Fan, H.F. (1997). A computer program to derive (3+1)-dimensional symmetry operations from two-line symbols. *J. Appl. Cryst.,* **30**, 73-78.

Gelder, R. de, Randy, I., Lam, E.J.W., Beurskens, P.T., Smaalen, S. van, Fu, Z.Q. & Fan, H.F. (1996). Direct methods for incommensurately modulated structures. On the applicability of normalized structure factors. *Acta Cryst.* A**52**, 947-954.

Hao, Q., Liu, Y.W. & Fan, H.F. (1987). Direct methods in superspace I. Preliminary theory and test on the determination of incommensurate modulated structures. *Acta Cryst.* A**43**, 820-824.
Hauptman, H. & Karle, J. (1953). Solution of the phase problem. I. The centrosymmetric crystal. ACA Monograph No. 3. Dayton, Ohio: Polycrystal Book Service.
Janssen, T., Janner, A., Looijenga-Vos, A. & de Wolff, P. M. (2006). Incommensurate and commensurate modulated structures. *International tables for crystallography*, Vol. C, ch. 9.8. pp. 907-955, edited by E. Prince, Springer.
Karle, J. & Hauptman, H. (1956). A theory of phase determination for the four types of non-centrosymmetric space groups 1P222, 2P22, 3P12, 3P22. *Acta Cryst.* **9**, 635–651.
Li, X.M., Li, F.H. & Fan, H.F. (2009). A revised version of the Program VEC. *Chinese Physics B* **18**, 2459-2463.
Mo, Y.D., Fu, Z.Q., Fan, H.F., van Smaalen, S. Lam, E.J.W. & Beurskens, P.T. (1996). Direct methods for incommensurate intergrowth compounds III. Solving the average structure in multidimensional space. *Acta Cryst*. A**52**, 640-644.
Sayre, D. (1952). The squaring method: a new method for phase determination. *Acta Cryst.* **5**, 60-65.
Sha, B.D., Fan, H.F., van Smaalen, S., Lam, E.J.W. & Beurskens, P.T. (1994). Direct methods for incommensurate intergrowth compounds II. Determination of the modulation using only main reflections. *Acta Cryst*. A**50**, 511-515.
Smaalen, S. van (1995). Incommensurate crystal structures. *Crystallogr. Rev.* **4**, 79–202.
Wan, Z.H., Liu, Y.D. Fu, Z.Q., Li, Y., Cheng, T.Z., Li, F.H. & Fan, H.F. (2003). Visual computing in electron crystallography. *Z. Krist.* **218**, 308-315.
Wolff, P. M. de (1974). The pseudo-symmetry of modulated crystal structures, *Acta Cryst*. A**30**, 777-785.
Wolff, P. M. de, Janssen, T. & Janner (1981). The superspace groups for incommensurate crystal structures with a one-dimensional modulation. *Acta Cryst*. A**37**, 625-636.
Yamamoto, A. (1982). Structure factor of modulated crystal structures, *Acta Cryst.* A**38**, 87-92.
Yamamoto, A. (1996). Crystallography of quasiperiodic crystals. *Acta Cryst*. A**52**, 509–560.
Yao, J. X. (1981). On the application of phase relationships to complex structures XVIII. RANTAN—Random MULTAN. *Acta Cryst.*, A**37**, 642-644.

Editor's note: following appendices deposited as a separate zip file with this newsletter.

Appendix I. Source codes of the first edition of DIMS (1994)

Appendix II. Source codes of SPGR4D

Appendix III. Source codes of SYMBOL1to2

Appendix IV. Source codes of DIMS (2000) for MS Windows

Appendix V. Source codes of DIMS (2009) for Linux

About Crunch 1.5 Direct Methods Software

R. de Gelder, R.A.G. de Graaff & W.J. Vermin
BFSC Leiden Institute of Chemistry, Gorlaeus Laboratories, PO Box 9502, 2300 RA Leiden, The Netherlands. E-mail: rag@chem.leidenuniv.nl ; r.degelder@science.ru.nl ; WWW: http://www.bfsc.leidenuniv.nl/software/crunch/

1. Introduction

Crunch uses the concurrent maximization of the determinants of small Karle- Hauptman matrices to get to phases that bear some relation to the structure that is being looked for. For the development of this concept the reader is referred to the papers mentioned below:

On the construction of Karle-Hauptman matrices. Acta Crystallographica A46 (1990), 688-692, R de Gelder, R.A.G. de Graaff & H. Schenk. X-ray Department Gorlaeus Laboratories, PO Box 9502, 2300 RA Leiden, The Netherlands

Automatic Determination of Crystal Structures using Karle-Hauptman Matrices. Acta Crystallographica A49 (1993), 287-293, R. de Gelder, R.A.G. De Graaff & H. Schenk. X-ray Department Gorlaeus Laboratories, PO Box 9502, 2300 RA Leiden, The Netherlands

and

On the automatic extension of incomplete models by iterative Fourier calculation. Journal of Applied Crystallography 17 (1984), 364-366, A.J. Kinneging & R.A.G. de Graaff. Department of Biophysical Structural Chemistry, PO Box 9502, 2300 RA Leiden, The Netherlands

Looking at Crunch, there are fundamentally three stages which must be considered. All are discussed in the papers quoted.

1. Checking the input, calculating E's if necessary and the construction of a set of matrices containing a suffcient number of independent reflections to be phased. (Crunch.uni, thinkc and deter in Cruncher.uni)

2. Maximizing the product of the determinants of the matrices obtained as a function of the phases, starting from some point in reciprocal space. (pmf and deter, Cruncher.uni)

3. Evaluating the phaseset obtained by trying to find a complete model based on the map obtained by Fourier transformation of the phaseset (autofour, Cruncher.uni)

Re 1. At least N reflections should be phased where N is not less than twice the number of atoms to be found. This is why more than one matrix are needed. Deter constructs the matrices, maximizing the average E-value of the reflections present in the matrices. Also care is taken to ensure that reflections and their symmetry equivalents occur in the matrices a suffcient number of times. Thinkc determines whether the right input to deter, specifying the matrix to be used in phase refinement, has been found.

This section of Crunch is governed by the shell scripts Crunch.uni and Cruncher.uni. Crunch.uni also reads the reflection data, calculates E-values and generally sets the stage for the structure determination iterations which are carried out by the script Cruncher.uni. Both scripts are given in an appendix to this paper.

Re 2. This is an extension of the maximum determinant rule first formulated by Tsoucaris (Tsoucaris, 1970). Pmf calculates Patterson compliant starting sets, which according to Sheldrick provide a better starting point to the phase determination process. The maximum determinant rule states that the most

probable set of phases of the reflections present in a Karle-Hauptman matrix maximizes the value of the determinant of the matrix. In deter the maximization is done following the method described in de Gelder, de Graaff and Schenk, 1993.

Re 3. The reflections present in the matrices represent a non-random selection from reciprocal space. This means that direct Fourier transformation after phasing usually produces a map which is not a good approximation of the true map. False peaks are present and many atoms may be missing. Autofour (Kinneging and de Graaff, 1984) takes care of this if the phases found by deter are good enough.

2. Phase refinement

In this section the source text of the key routine is presented, together with an explanation of what happens where and when.

```
      module modopdet
      use modunit
      use moddet1
      use modjaap1
      use modschuif
      implicit none
      contains
      SUBROUTINE OPDET(NHKLF,EMX,PMX,IJPA,IJQA,DETN,ICYCMX)
      implicit none
      integer,                    intent(in)    :: nhklf
      real,    dimension(:,:,:), intent(inout) :: emx,pmx ! (nm,nm,nnrm)
      integer, dimension(:,:),   intent(in)    :: ijpa    ! (7,nfs)
      integer, dimension(:,:),   intent(in)    :: ijqa    ! (8,nijqa)
      real*8,                    intent(out)   :: detn
      integer,                   intent(in)    :: icycmx

!
!  OPDET MAXIMIZES THE DETERMINANT
!  SEE ACTA CRYSTALLOGRAPHICA
!  LAPACK ROUTINES ZPOTRF, ZPOTRI ARE USED TO FACTORIZE AND INVERT
!  KH MATRIX
!
!  NNRM DETERMINANTS ARE MAXIMIZED CONCURRENTLY
!  ONLY REFLEXIONS WITH E>ELIM ARE CONSIDERED BY OPINIT
!
!  PARAMETERS
!
!
!  EMX(NM,NM,NNRM) = INPUT, output =
!  E-VALUES IN THE LOWER TRIANGLE
!
!  PMX(NM,NM,NNRM) = INPUT, OUTPUT =
!  INPUT: START PHASES IN RADIANS
!  OUTPUT: REFINED PHASES IN RADIANS
!
!  NM = INPUT =
!  DECLARED DIMENSIONS OF THE MATRICES
!
!  AMX(NM,NM,NNRM)
!  SCRATCH ARRAY, AFTER DET1, JAAP1 HAVE BEEN CALLED LOWER TRIANGLE CONTAINS
!  THE INVERSE OF KH MATRIX, THE UPPER CONTAINS ORIGINAL VALUES STILL
!
!  IJPA,IJQA: = INPUT = SEE OPINIT
!  W(NFS,7): SCRATCH
!  FAS: SCRATCH PHASES
!  GRAD: SCRATCH GRADIENTS
!  FDET(NNRM): SCRATCH, INDIVIDUAL DETERMINANT VALUES FOR EACH MATRIX
!
```

```
      COMPLEX*16, dimension(:,:,:), allocatable :: AMX
      real*8,     dimension(:),     allocatable :: fdet,grad
      real,       dimension(:),     allocatable :: fas
      real,       dimension(:,:),   allocatable :: w

      integer NNRM,NM,NIJQA,NFS,ifail
      INTEGER CONV,DIAG
      DIMENSION DT(9)
      REAL*8 V,VERM,DETO
      integer mk,nc,ma,lk,iry,iko,ifud,icycs
      integer j,icyc,icv,i,l
      real x,vfud,twopi,thpid2,pid12,pid2,dt
      real z,y,pi,hoek,fudmin,gstop,fudmax,dfud
      logical converged

!
!  DIAG = NUMBER OF TIMES THE ELEMENTS HAVE BEEN DECREASED BY 10%
!
!  THE PHASES
!
!     DATA IFAS /6/
!
!  THE FUDGE FACTORS
!
      DATA IFUD /6/
!
!  W(*,1-3)  SHIFTS OF THE LAST THREE ROUNDS
!
!  THE GRADIENTS
!
!     DATA IGRAD /4/
!
!  STORAGE OF PHASES FROM LAST CYCLE
!
      DATA ICV /5/
!
!  MAXIMUM OF ANY FUDGE FACTOR
!
      DATA FUDMAX /3.0/
!
!  MINIMUM OF ANY FUDGE FACTOR
!
      DATA FUDMIN /.01/
!
!  FUDGE FACTORS ARE MULTIPLIED BY VFUD IF DETERMINANT INCREASES
!
      DATA VFUD /1.1/
!
!  FUDGE FACTORS ARE DIVIDED BY DFUD IF NOT
!
      DATA DFUD /1.5/
!
!  THE MAIN DIAGONAL OF THE MATRIX IS MULTIPLIED BY V IF DET <0
!
      DATA V /0.9/
!
!  REFINEMENT STOPS IF THE CHANGE IN THE VALUE OF DET IS SMALLER THAN
!  GSTOP OVER THE LAST 10 CYCLES
!
      DATA GSTOP /0.01/
!     DATA GSTOP /0.005/

      nnrm = size(emx,3)
      nm   = size(emx,1)
      nfs  = size(ijpa,2)
      nijqa = size(ijqa,2)
```

```
      allocate(amx(nm,nm,nnrm))
      allocate(fdet(nnrm))
      allocate(w(nfs,7))
      allocate(fas(nfs))
      allocate(grad(nfs))

      PI     = 4.0*ATAN(1.0)
      TWOPI  = 2.0*PI
      PID12  = PI/12.0
      PID2   = PI/2
      THPID2 = 3*PID2

      VERM   = 1.0
      ICYCS  = 0
      DIAG   = 0

      nfs_eq_0: IF(NFS.eq.0) then
        WRITE(LO,*)'NHKLF=  ',NHKLF
        IF(NHKLF.EQ.1)THEN
          CALL DET1(EMX,PMX,AMX,
     1     IJPA,IJQA,FAS,DETO,FDET,IFAIL,VERM)
          IF(IFAIL.EQ.0)THEN
            WRITE(LO,3)DETO
     3      FORMAT(' OPDET: WITHOUT REFINEMENT THE DETERMINANT IS: ',
     1             E12.4)
          ELSE
            WRITE(LO,4)
     4      FORMAT(' OPDET: PHASES PUT IN RESULT ',
     1             'IN A NEGATIVE DETERMINANT')
          ENDIF
        ENDIF
      else

        convloop: do conv=0,1
          converged = .false.
          if(conv .eq. 1) then
            WRITE(LO,*)
     1      ' OPDET: NOW REFINE WITH SPECIAL REFLEXIONS RESTRICTED'
          endif
          DO I=1,NFS
!
!  GET ROW AND COLUMN OF VARIABLE I
!
            IRY=IJPA(1,I)
            IKO=IJPA(2,I)
            MA=IJPA(3,I)
!
!  CORRECT RESTRICTED PHASE
!
            Z=PMX(IRY,IKO,MA)
            MK=IJPA(6,I)
            IF(MK.ne.1) then
              Y=(MK-1)*PID12
              X=AMOD(Z-Y,TWOPI)
              IF(X.LT.0.0) X=X+TWOPI
              IF(X.GT.PID2.AND.X.LT.THPID2) Y=Y+PI
              Z=Y
            endif
            Z=AMOD(Z,TWOPI)
            IF(Z.LT.0.0) Z=Z+TWOPI
            FAS(I)=Z
          enddo
          LK=0
          VERM=1.0
!
```

```
!  CALCULATE DETERMINANT
!
       mainloop: do
         calcdet: do
           DO I=1,NFS
             W(I,IFUD)=0.3
           enddo
           DO MA=1,NNRM
             DO I=1,NM
               EMX(I,I,MA)=1.0
             enddo
           enddo
           CALL DET1(EMX,PMX,AMX,
     1               IJPA,IJQA,FAS,DETO,FDET,IFAIL,VERM)
           IF(IFAIL.EQ.0) exit calcdet
           IF(LK.EQ.1) exit convloop

           DIAG=DIAG+1
           VERM=VERM*V
         enddo calcdet

         WRITE(LO,1) DETO
    1    FORMAT(' OPDET: INITIAL VALUE OF THE DETERMINANT=',E20.10)
         ICYC=1
         NC=0
         cycleloop: do
!
!  STORE PHASE IN W
!
           DO I=1,NFS
             W(I,ICV)=FAS(I)
           enddo
!
!  CALCULATE INVERSE MATRIX
!
           CALL JAAP1(AMX,IJPA,IJQA,GRAD,FDET)
!
!  THE LOWER TRIANGLES OF AMX CONTAIN THE INVERSE OF AMX
!  NOW CALCULATE SHIFTS WITHOUT FUDGEFACTORS
!

           CALL SCHUIF(AMX,W(:,1),W(:,ICV),GRAD,IJPA,
     1                 IJQA)

           loop: do

             if ( .not. any(w(1:nfs,ifud) .gt. fudmin)) then
               exit cycleloop
             endif

             DO I=1,NFS
               MK=IJPA(6,I)
               IF(MK.eq.1.or.CONV.ne.1) then
                 HOEK=W(I,ICV)+W(I,1)*W(I,IFUD)
                 FAS(I)=AMOD(HOEK,TWOPI)
                 IF(FAS(I).LT.0) FAS(I)=FAS(I)+TWOPI
               endif
             enddo
!
!  NOW RECALCULATE DETERMINANT
!
             CALL DET1(EMX,PMX,AMX,
     1          IJPA,IJQA,FAS,DETN,FDET,IFAIL,VERM)
             if (ifail .eq. 0 .and. detn .gt. deto ) exit loop
!
!  DETERMINANT NEGATIVE OR SMALLER THEN THAN OLD VALUE
```

```
!
                DO I=1,NFS
                  W(I,IFUD)=W(I,IFUD)/DFUD
                enddo
              enddo loop

              NC=MIN0(10,NC+1)
              if(nc .ge. 10) then
                X=(DETN-DT(1))/DETN
                IF(X.LT.GSTOP) exit cycleloop
                DO I=1,8
                  DT(I)=DT(I+1)
                enddo
                NC=NC-1
              endif DT(NC)=DETN if (icyc .ge. icycmx) then
                write(lo,*)' OPDET: TOO MANY CYCLES'
              else
                ICYC=ICYC+1
                ICYCS=ICYCS+1
                DETO=DETN
                DO I=1,NFS
                  W(I,IFUD)=AMIN1(W(I,IFUD)*VFUD,FUDMAX)
                enddo
              endif
            enddo cycleloop

            L=0
            IF(DIAG.GT.0) THEN
              LK=1
              DIAG=DIAG-1
              L=1
            ENDIF ! wwvv in original, this endif was placed after next endif
            IF(DIAG.EQ.0)THEN
              VERM=1.0
            ELSE
              VERM=VERM/V
            ENDIF
!
!  MAIN DIAGONAL HAS BEEN INCREASED, DECREASE AND REFINE AGAIN
!
              if (l .eq. 0) exit mainloop
          enddo mainloop

          converged = .true.
        enddo convloop

        if (converged) then
          WRITE(LO,*) ' OPDET: CONVERGENCE OBTAINED'
        else
          WRITE(LO,*)
     1     ' OPDET: NO CONVERGENCE, MATRIX IS NOT POSITIVE DEFINITE'
        endif

        WRITE(LO,10001) ICYCS,DETN
10001   FORMAT(' OPDET: NUMBER OF CYCLES =',I5,' DETERMINANT=',
     1           E20.10)
        DO MA=1,NNRM
          DO I=1,NM
            DO J=1,I
              PMX(I,J,MA)=AMOD(PMX(I,J,MA),TWOPI)
              IF(PMX(I,J,MA).LT.0.0) PMX(I,J,MA)=PMX(I,J,MA)+TWOPI
            enddo
          enddo
        enddo

      endif nfs_eq_0
```

```
deallocate(amx)
deallocate(fdet)
deallocate(w)
deallocate(fas)
deallocate(grad)

RETURN
END SUBROUTINE opdet
end module modopdet
```

3. Model extension

The program autofour, run by Cruncher.uni, evaluates and tries to extend the model obtained after phase refinement. The program uses iterative Fourier calculations to improve and extend the model. The individual atoms in the model are checked continuousy using a critrion based on R2 Wrongly placed atoms are deleted from the model as soon as is possible, while the quality of the model is checked against the expected value of the criterion mentioned earlier. For a description of the procedure see Kinneging and de Graaff, 1984.

4. The scripts

The script crunch.uni is prepared during the installation of Crunch. Crunch.uni governs the iniial stages of the procedure as well as the communication with the user. E-values may be calculated in two different ways, things like cell contents and other input parameters are checked and stored. See for more information the manual which is appended..

Editor's note: scripts are is listed starting on page 339 within Addendum B: Computing Software manuals and reference materials.

5. The manual

Crunch is a direct methods program developed for solving difficult small and medium - sized structures ab initio. Datasets should be of atomic resolution. The system has been designed to run under the Unix operating system. The current version has been tested on various multi-processorsystems using the Linux operating system or OS X 10.4 or better. Although Crunch is aimed in principle at difficult equal-atom problems, the program may be used effectively for routine structure determination of heavy-atom structures too. The program needs about 20 Mb disk space. If supported the program will use OpenMp and Gpu to speed up calculations. Memory allocation is dynamic and automatic.

The system consists of two main programs and assorted utilities. The first pro- gram, deter, determines the phases. The second program, autofour, evaluates the results of each deter cycle, trying to find the complete model based on the results obtained by deter. The contents of the unit cell should be given as accurately as possible. Especially the second principal program in the system, the section which tries to find the complete model based on the results of the first section, depends on the availability of reasonable estimates of B-overall and the scale. Most matrix and vector manipulations are done using the Blas and Lapack routines. If you do not have gfortran, or intel fortran + numerical library available on your machine it is of advantage to use processor specific optimized versions of this software. On Mac Os X systems use the libraries supplied by Apple. Usually the configure software used in the installation will take care of all this by setting the correct fiags. Makefiles are supplied for the program system. An include file 'fiags' which defines the fiags used in the makefiles is prepared during installation. This file may be adapted to the local situation.

Crunch supports just about any input of structure factors you can think of as long as the files contain hkl, F or F**2 and sigma(F or F**2), one reflection pro record and no records containing other items. The

entries in the file must be separated by blanks, comma's or plus or minus signs. A record such as 12 3 4 1.2340.566 will not do. Crunch 1.1 or higher no longer requires the presence of the Dirdif system (inquiies Dirdif to Rene de Gelder)

In earlier versions of Crunch Dirdif was used to handle the crystallographic data such as cell contents, unit cell dimensions, symmetry etc. Crunch uses a reflection file containing the asymmetric unit in reciprocal space ONLY. However, your refiection file is cleaned of redundancies automatically. Atomic coordinates are produced in the .pdb and .spf formats. Visual inspection of the results is therefore straightforward, using the Pluton/Platon graphic suite written by Ton Spek (email: a.l.spek@chem.uu.nl). Unix scripts are written to be compatible with the Bourne/Korn shells. The scripts provided do not allow the running of more than one crunch-job from within one directory. However, multiple runs of Crunch on one computer from different directories and/or by different users are allowed.

Crunch was developed for the ab initio solution of the phase problem, which also covers the situation where a crystallographer doesn't know anything about his compound except that it is organic and its approximate atomic weight. This has proven to be extremely useful for the identification of unknown natural compounds.

If you've used Crunch in any resulting paper please refer to:

Automatic Determination of Crystal Structures using Karle-Hauptman Matrices. Acta Crystallographica A49 (1993), 287-293, R. de Gelder, R.A.G. De Graaff & H. Schenk. X-ray Department Gorlaeus Laboratories, PO Box 9502, 2300 RA Leiden, The Netherlands

6. Installation

Copy crunch1.5.tgz into the directory where you would like to have Crunch installed. Next untar the archive by typing 'tar zxvf crunch1.5.tgz'. Follow with 'cd crunch1.5' and './configure'. Next type 'make clean' followed by

'make dep' and 'make'.

Crunch1.5 is being installed now

Mind that suffcient (> 20Mb) disk space is available before you start the installation. You need to have gunzip, the gnu unzipper, available on your system as well as a fortran90 compiler. Gfortran springs to mind. Another good choice is the Intel ccmpiler system. The configure step takes care of the compiler choice. OpenMp and Gpu calculation are supported. In the directory you've chosen for Crunch to be installed into, a new directory 'crunch1.5' should now be present. This directory should contain the following: A directory manual containing this manual - in the form of the files manual.aux, manual.toc and manual.tex - the files runit, fiags and the directories drivers, programs, source, pytlud and rn001. Runit is a dynamic link to the script which actually runs the program. Move this to some suitable directory in your PATH, such as /usr/local/bin and chage it as required. Copy the directories `rn001` and `pytlud` into the directory where you usually solve your structures. You may have to close and reopen your X-terminal to activate the link 'runit'.

You are now ready to proceed.

The configure script checks for the presence of a Fortran compiler on your system.

The file configure.log contains details on the compilers etc. which are going to be used for the installation. The prefered option on Linux and OS X systems is GNU's gfortran. If this is not present I

strongly advise you to get it installed on your system. For me this compiler works on Intel Macs too even though the current version is experimental.

7. Starting the first time

Open the directory rn001 which you have just created. Type `'crunch rn001 clear'`. This will leave you with just the files rn001.crysin, rn001.frefa and crunch.log. Now run Crunch by typing `'crunch rn001 try 1 5'`. Crunch will now proceed to solve the structure. On a modern computer this should take just a few seconds. The space group is P21, N=48, it is a simple steroid, an equal-atom structure.

When asked for input, choose default options by giving a return. When the program is finished your directory rn001 should contain the files rn001.report, rn001.pdb and rn001.spf, among others. Read the report file. If you wish to do so, plot the result using Ton Spek's program Platon, using the rn001.spf file. Compare the results, checking against the files present in the rn001 directory supplied. If you meet with any problems please contact the authors.

8. Solving a structure of your own

Open the directory where you would like to solve your structure. In the following 'code' stands for the name of your compound. This may be any suitable abbreviation, of course. If you already have a file in there 'code.crysin', containing the crystallographic data in the Dirdif format, so much the better. If not, Crunch will prompt you for the information required. The file 'code.crysin' will be created.

The file 'code.crysin' contains the usual crystallographic data such as cell constants, wavelength used, cell contents etc. It is important to give the cell contents as accurately as is possible. Next, you need a file containing h,k,l, F and sig(F), one reflection for each record, with proper separation between numbers (spaces, comma's or a plus or minus sign are required). The values for h,k,l should be consecutive on your file. However, the other variables may occur in any order. E.g. the hkl may be the last three numbers etc. In the record other numbers may be present, the program will skip them if required. Before you do anything else your data file must be converted into a 'code.frefa' file. Type `'crunch code conhkl 'filename'`. Here 'filename' is the full name of the file - NOT the full path - containing your reflections. Crunch will ask you a few questions and next the file will be converted to a Crunch-friendly one. The whole procedure is self explanatory. Now type `'crunch code try 1 5'`. Provide the information you are asked for.

Use the defaults indicated. Crunch determines automatically whether to treat the problem as a heavy atom structure or an equal atom one. In some cases you are given the option to treat for instance an organic compound containing chlorine as an equal atom problem. Basically you just give return a few times, Crunch will use sensible defaults and your structure will be solved. Try larger values of the last number given - 5 in this case - if you expect the solution to be difficult. Your structure should come out if all goes well. Crunch is a multi-solution method. In the example given, 5 random starts will be used to find the solution.

If you suspect your data is not truly of optimal quality, Crunch works best if you calculate E-values from your reflections using the option 'blessing'. See for details the section on syntax(Nr 7).

9. Using pmf

George Sheldrick, the author of the Shelx Program series, has pioneered the use of Patterson compliant starting sets instead of just random atomic positions to begin with. Crunch supports this as well. Use `'crunch code pmf 1 5'` to generate this type of start for Crunch. Next a normal run will use the starts generated by pmf automatically. The number of trials is limited by the number generated by pmf.

See Crunch syntax for further detail. Note that using pmf is possible only if Blessings E's have been calculated. The program stops if this is not the case, telling you why.

10. A few helpful hints

1. Sometimes heavy atoms are disordered, for instance the chlorine in a per- chlorate may be quite 'mobile'. If you expect something like that treat your structure as an equal atom one. Sometimes changing the cell contents – in 'code.crysin' - isfseeded in order to do this: a) If Crunch does not give you the option or b) you have a largeish organic molecule containing one or two sulfur or chlorine atoms. If Crunch has problems fiing a solution, even if you told the system it is dealing with an equal atom problem, change the cell contents in your code.crysin file, replacing the heavier atoms by an equivalent number of oxygen atoms.

2. Atoms such as P, S and Cl are heavy in the context of Crunch. hat is compared to first row elements such as C and O etc.

3. If you have a transition metal or an even heavier element in your compound DO NOT specify Cl and the like as heavy atoms.

4. If something has gone wrong, e.g. you've started Crunch while you didn't mean to and next interrupted the proceedings by 'Control c', ALWAYS start again by using the Crunch option 'clear': Type `'crunch code clear'`. This will clean up your directory, preventing otherwise inexplicable failures.

NB using the option `'clear'` will delete the E-values you may have calculated using the option `'blessing'`

11. Crunch' syntax

You may use any one of the options given below running the program. Always type your alias or the name of the dynamic link to the Crunch-script, followed by the complete specification of the run. The general syntax of a Crunch command line:

```
crunch code run [start end] [first n, last n, all]
```

Below the variables 'crunch', 'code' and 'run' have the following meaning through- out:

crunch: The name suggested for the link to or the alias of the script crunch.uni. In the distribution this link is named runit

code: compound name (e.g. rn001 in the test structure).

run: Defines the type of run which will be executed.

start: The number of the first trial do be done.

end: The number of the last attempt.

The last two values must be specified if one of the options 'first' or 'last' is to be used.

`conhkl filename` This type of run has a syntax different from the others. Use this run to convert your local structure factor file into a crunch-friendly (.frefa) format. The filename should be given with extension(s). See for more information the section Examples.

`clear` Most files created by earlier runs of Crunch are removed. Only the .crysin, deter.use, autofour.use, pmf.use and .frefa files are kept.

`blessing` E-values are calculated and saved for further use. The routines from the DREAR-suite, Levy and Eval are used. Reference: Blessing, R.H. & Smith, G.D. (1999). "Difference structure factor normalization for heavy-atom or anomalous scattering substructure determinations" J. Appl. Cryst. 32, 664-670. Until the option 'clear' is used to clean up, these E-values will be used rather than the ones calculated by Normal.

`pmf` If you have first calculated E's using the option blessing, the option pmf creates a file 'coordinates' containing partial structures derived from the patterson function. These will be used as starts to the phasing process instead of the random atomic positions which are used normally. Values of start and end must be given to determine the number of starts to be generated.

`try` Crunch uses random starts from 'start' until a solution is found or 'end' is reached. If the structure is found various files such as code.pdb, code.spf and code.report are created. Some information on all trials done is given in the file 'hits'.

`collect` Scan all crunch-attempts from 'start' to 'end'. The results are given in the file 'hits'.

`deter` Only the section calculating the phases - deter - is executed. The results are saved in files 'code.phi1...n'. As above, the calculations are done for the trials 'start' to 'end'.

`peaks` The phases generated using the option 'deter' are converted into possible fragments, using Exfit. This again is done for the trials 'start' through to 'end'.

`recycle` This option implements a sort of 'shake and bake' strategy. The trial 'start' is run in the normal way. If a solution is obtained then the program stops as usual. If not, the best model obtained in the autofour section is used to calculate phases to be refined using deter. A new model is found etc. This process is continued until the structure is solved or 'end-start+1' cycles are completed.

`autofour` This option may be used in two completely different ways:

1. You may have found a heavy atom or a small part of the structure by using some other method. If you then prepare a .frefa file as was described above, followed by creating a file `'code.pek'` containing the known part of your struc- ture, you can try to extend the model to the complete structure by just typing `'crunch code autofour'`, not giving the values start and end. A description of the code.pek file is given below.

2. You have created files `'code.pek1...n'` by using the peaks option. Then type `'crunch code autofour start end'` to try to extend one or more of the models obtained.

The values `'start'` and `'end'` are optional except if you use `'recycle'`, `'deter'`, `'pmf'` or `'peaks'`. The default values are 1 and 10 respectively.

Using the option `'try'`, which is normal when you are trying to solve a structure, usually the second step, model extension using autofour, is started with the largest consistent fragment present in the map. However, following the value of 'end' you may specify the options `'first'` or `'last'`, followed by a

number 'n', or 'all'. The effect is that the first 'n', the last 'n' or all peaks found in the map, are used as input to autofour. Remember to specify the values 'start' and 'end' if you'd like to use the options just described. The script gets confused if you specify just the parameters 'first' or 'last', interpreting them as values for 'start' and 'end'.

An example run: 'crunch rn001 collect 27 74' means crunch rn001 and collect 'hits' from sets 27 to 74. Hits contains a brief report on this run. If you are using the option 'try' the results of the first successful run are summarized in a file code.report. Should you want to look at the results of a particular successful run collected in the file hits, use the command 'crunch code try n n'. Further examples may be found in the designated section.

12. Required before running Crunch

You just need a structure factor file, the usual crystallographic data such as cell constants, cell contents, the wavelength used etc. Any structure factor file containing one reflection pro record will do, as long as h,k,l, Fobs (or F**2) and sigma(F) (or sig(F**2)) are there. Remove any records which contain "non intensity" information from the file. Convert the file to a 'code.frefa' file using the option conhkl. Do use an alias or the dynamic link provided to run the script crunch.uni. In the special case you want to use the syntax 'crunch code autofour', that is, you want to use crunch just to extend a model found some other way, you need to prepare one more file, 'code.pek'.

13. What to do in the case of problems?

In the case of failure of a standard run of Crunch - the structure does not come out - try changing some input parameters to deter. Look in the file deter.asc which should be present in the Crunch subdirectory 'drivers'. Create a file called deter.use in the directory you are trying to solve your structure from. This file should be of the same format as deter.asc, however you need to specify only the items you wish to change.

1. Increase the number of trials.

2. Use the option 'blessing' to calculate E-values and try to solve the structure again. This is particularly useful if the quality of your data is not completely up to scratch. In one test, collecting the results of 100 trials, using Blessing's E's six solutions were found, instead of two obtained using Normal80's E's.i

3. Try generating starts using pmf

4. In the subdirectory 'drivers' of the Crunch system files 'deter.asc, autofour.asc and pmf.asc' are present. These files contain defaults for the programs deter, dutofour and dmf. In the directory were you are trying to solve your structure create a file 'deter.use', of a similar format. You need to supply only thoe parameters you would like to give non-default values to. It is best not to change NNRM, INMX and NINMX in this way, as Crunch allows you to specify the number of matrices used and their orders when you start your efforts or after you've cleaned up the directory.

Try different values for the following parameters:

IBK the default value is calculate by the system, based on Crunch usng approximately 256 Mb. However, sometimes smaller values give better re- sults. Try something like 1500.

NRL Increase or decrease the value a bit.

ICRA You might like to try the value 2 here.

NOTF This parameter defines the weight of forbidden refiections in the matrix construction process. The default is one, i.e. they are treated as valuable compared to very weak reflections. Sometimes treating them as statistically very weak works better: Change the value into zero. Changing the value into two is another option, increasing their importance.

Try all this using the option `'crunch code deter 1 1'`. Inspect deter.cout and try to maximize the average E-value in the matrices and/or the number of strong, independent reflections.

5. For larger structures it is sometimes useful to use matrices of smaller dimen- sions than the default chosen by Crunch. Type `'crunch code clear'`. Next, type `'crunch code deter 1 1'` and now, when the system asks if you would like to use default values for the matrix construction, type `'u'`.

Choose a different, preferably lower, number of matrices and/or a difierent order for the matrices. The minimum number allowed is 1. The number of matrices you may use is prescribed, not just any positive number will do. If you type a number here which is unacceptable Crunch will substitute the nearest acceptable one. The order of the matrices you may choose to be any odd number. Try to choose your parameters to fit the requirements. E-average should be high. The number of strong, independent reflections should be at least about 2.5 times the number of independent atoms you are looking for.

A significant number of symmetry equivalents - including Friedels - should be present. If you succeed in finding a set of matrices which looks promising, - high average E, enough strong independent reflections while containing a suffcient amount of redundancy - try to solve the structure again using the options `'try'` or `'collect'`.

6. Generate about a hundred solutions. Look in the file 'hits' for the solution which yields the lowest value for R2. Type: `'crunch code recycle n n+10'` or `20`. n is the number of the trial chosen. If this does not work and other trials looking better than most others exist, try them in the same way.

7. As there is a file deter.asc, so files autofour.asc and pmf.asc are present in the directory 'drivers'. The format is the same as is its use: You may override the values given in *.asc by creating a file *.use in your working directory, specifying different values therein. If you suspect your reflection data is not of terribly good quality, try a different value for STOPR, something like 40. STOPR is the threshold value of R2 in percentage points. For values of R2 below STOPR autofour considers the structure to be solved. Should your file contain reasonable standard deviations, try using IWEIGHT=1. Or, if you're trying to solve a structure containing one or two relatively heavy atoms such as Cl or S, try giving IVAL some value, say 10 or 20. The latter is a good idea too if you use Crunch just to extend a small model based on a few (heavy) atoms.

If all this does not work give up. Crunch will not solve this structure, at least not for you. If you like, contact the authors, giving full details of what you have done so far.

14. The file code.pek

For each atom in your input model the file should contain a record as defined below.

1 - 10		May be used for atom name	Format: 10X
11 - 20	FI	Table number belonging to this atom	Format: F10.0
21 - 30	X	The fractional x-coordinate	Format: F10.3
31 - 40	Y	The fractional y-coordinate	Format: F10.3
41 - 50	Z	The fractional z-coordinate	Format: F10.3

Note: The records specify the format of the code.pek file, needed if you want to run autofour with a model found NOT using Crunch. Pay attention to the parameter FI, its value should tally with the input

you have specified in your file 'code.crysin' For instance, you've found a Cl-atom and you would like to use this to find the other atoms. You specified Cl as the second atom type in your crysin file. Use FI=2.0 in this case.

15. Some useful files

`flags` This file contains the compiler and linking options during used during installation of the system. The adventurous user might want to play with this.

`code.frefa` The reflection file which must be present. However, you may convert a reflection file in a local format using the option provided. The .frefa file may be used in the Dirdif system as well.

`code.crysin` The Dirdif system file which contains the crystallographic data on your compound. This may be prepared interactively.

`hits` For each random start this gives the determinant value obtained as well as the values of R2 and R2 expected found by autofour.

`crunch.log` Contains a log of all Crunch's runs submitted from this directory. deter.use Similar to 'drivers/deter.asc'. Values specified will override defaults in 'drivers/deter.asc'.

`autofour.use` As 'deter.use' but defining the input to autofour.

`deter.cout` The output of the program deter. Only the output of the last run is kept.

`pmf.use` As before, you may change parameters for pmf this way.

`autofour.cout` The output of autofour's last attempt.

The last two files are not present if the Crunch run was successful. The following files are present in the case of success only.

`code.report` This file contains relevant information on the proceedings leading up to the result.

`code.spf` The coordinates in spf format. This file may be used to prepare plots of the molecule(s) found using Pluton or Platon.

`output` The file contains all output generated during the last successful attempt.

`code.pdb` The coordinates in Protein data bank format.

16. Examples

1. Converting a local structure factor file named 'xtal.dat'

In this example the user typed 'crunch hornef conhkl xtal.dat'. The following information appeared on the screen - the bit in [] was provided by the user - :

```
==========================================
**  CONHKL   conversion of  reflection files
==========================================
**  The first record on your reflection-file is: HKL  0  -2  0  66.23596 1.00171  1  SKIP
== > You must now specify your record type using keywords
```

```
**  Example:  hkl skip fobs sigma
**  This means:
**  The first 3 numbers are h,k,l, the 4th should be skipped,
**  the next one  is Fobs and the last one is sigma(Fobs)
**  Typing f2obs instead of fobs tells the program your file
**  contains F**2 rather than F
== > Specify your record:   [skip  hkl   fobs sigma  skip]
=========================================
**  conhkl ended
```

Users of the package XTAL will recognize the record shown as typical input of ADDREF, the program in XTAL which reads in structure factors. Hornef is the name of the compound. The programs shows the frst record of the file, followed by an explanation on how to proceed. Here the records consist of a bit of text, h,k,l, Fobs, sigma(Fobs) and some other information which Crunch does not need. By typing the record between brackets the user tells Crunch to skip the frst item and to consider the next fve numbers as h,k,l, Fobs and sigma respectively. The rest of the record is skipped again. Use lower case characters here! At the end of the run the fle 'hornef.frefa' was prepared, ready for use by Crunch.

2. Trying to solve a structure the first time.

The user next typed 'crunch hornef try'. On the screen appeared:
Crunch interactive, always give return to choose defaults

```
CELLCO   C  92  H  72  O  20
```

The structure is an Equal atom structure

```
[D]efault  or  [U ]ser  specified  matrix construction?
```

The cell contents are shown on screen. Crunch has decided that hornef is an equal atom structure. The user gave a return on the next question as he wanted to try a default run of Crunch. In this case a maximum of 10 random starting sets is tried. If you next run Crunch you will not be asked these questions again. Should you want to change anything, for instance the way the Karle-Hauptman matrices are constructed, you will have to remove the file 'code.par', in this case hornef.par. You may choose to use the command `'crunch code clear'` instead.

3. The user wants to collect the results of a number of trials, to see whether there are any (non)solutions which look more promising than the others. See the section 'What to do in the case of problems?'. He typed: `'crunch hornef collect 1 25'`. After a few moments the Unix prompt reappears.

4. On completion the file 'hits' is inspected. Solution number 9 looks better than the others: A higher value of the determinant and a lower final value of R2 are the criteria to look for. The user now typed: `'crunch hornef recycle 9 15'`. Again after a few seconds the Unix prompt reappears. Crunch will do the calculation for trial number 9 again. However, instead of proceeding to trial 10 in the case of failure, the best model obtained is used to calculate phases which are used as starting values for the maximization of the determinants. The refined phases are used to calculate a new start for the second section of Crunch. In this case this is done six times (15-9). Of course Crunch stops iterating if a satisfactory solution is obtained. Note that it is not necessary to use the options `'try'` or `'collect'` before using `'recycle'`. It is perfectly valid to try to solve your structure using a run such as: `'crunch hornef recycle 1 10'`.

17. Download

The crunch webpage is at http://www.bfsc.leidenuniv.nl/software/crunch/

Experience converting a large Fortran-77 program to C++

Ralf W. Grosse-Kunstleve, Thomas C. Terwilliger, Paul D. Adams
Lawrence Berkeley National Laboratory, One Cyclotron Road, BLDG 64R0121, Berkeley, California, 94720-8118, USA. E-mail: RWGrosse-Kunstleve@lbl.gov

Introduction

RESOLVE is a widely used program for statistical density modification, local pattern matching, automated model-building, automated ligand-fitting, and prime-and-switch minimum bias phasing (Terwilliger 2000, Terwilliger 2003). In recent years it has been developed primarily in the context of PHENIX, which is a rapidly growing software suite for the automated determination of macromolecular structures using X-ray crystallography and other methods (Adams et al., 2002).

PHENIX is a "Python-based Hierarchical ENvironment for Integrated Xtallography". The main layers of the hierarchical organization are a Graphical User Interface (GUI) written in Python (http://python.org), Python scripts implementing applications such as structure solution and refinement, and C++ extensions for numerically intensive algorithms (Abrahams & Grosse-Kunstleve, 2003; Grosse-Kunstleve & Adams, 2003).

The origins of RESOLVE predate the PHENIX project. The original implementation language is Fortran-77. Since Fortran RESOLVE makes heavy use of global data it is not suitable for tight integration as a Python extension. Therefore, Python scripts write RESOLVE input files to disk and call RESOLVE as an external program which writes its outputs to disk. These are read back by the Python scripts to continue the automated processes.

Communicating data through the file system is increasingly problematic as the number of CPUs in a machine or cluster is steadily increasing. For example, if several hundred processes write large density-modified maps or reflection files to a network file system simultaneously, the I/O tends to become a bottleneck. In our development work we found it necessary to mitigate the I/O congestion by writing intermediate files to a local disk (such as /var/tmp).

Another general problem of systems starting external processes is the vulnerability to improper configuration or limitations of the shell environment, for example a very long PATH, or LD_LIBRARY_PATH leading to surprising conflicts. Unfortunately, such problems are reported regularly and they tend to be time-consuming to debug since, by their very nature, they depend heavily on the environment and are difficult to reproduce.

Another set of problems arises from the mixing of Fortran and C++. Using C++ libraries from Fortran is very difficult, although it is not impossible. Fortran RESOLVE uses the latest CCP4 libraries (Collaborative Computational Project, Number 4, 1994) including indirectly MMDB (http://www.ebi.ac.uk/~keb/cldoc/) which is implemented in C++; the required Fortran interface layer is provided by CCP4. A major disadvantage of the C++/Fortran mix is that the build process becomes much more complex than a pure C++ or pure Fortran build.

Mixing Fortran and C++ and starting external processes has a number of other practical drawbacks, for example the external process calls are different under Windows and Unix, and under Windows and Mac OS X it is cumbersome to install any Fortran compiler. Any particular problem by itself is relatively small, but in a system as large as PHENIX the small problems add up to a significant permanent stream of distractions hampering long-term progress. Therefore it was decided to make PHENIX as a whole more uniform - and by implication easier to maintain, develop and distribute - by converting the RESOLVE implementation from Fortran to C++.

Outline of the RESOLVE conversion work

Largely due to the size of the RESOLVE Fortran code, the route taken in the conversion is a very conservative one. The Fortran code is divided into about 700 files with ca. 85000 non-empty lines, of which ca. 72000 are non-comment lines. These sizes practically preclude rewriting RESOLVE as idiomatic C++ code from scratch. Even when producing new code at the very high sustained rate of of 100 lines per day, 50 five-day weeks per year, it would take almost 3 1/2 years to finish the work. After all this time there would be very little new still from a user's perspective. Thus, we wanted to find a more evolutionary approach that produces new results while gradually improving the underlying framework in order to accelerate future developments. Therefore we did not try to turn idiomatic Fortran into idiomatic C++, but instead concentrated our resources on the more limited task of mechanically converting Fortran syntax to C++ syntax that still resembles the original Fortran. The converted code can be evolved over time to increasingly take advantage of the much richer features of the C++ language.

The company Objexx (objexx.com) was hired to help with the mechanical conversion work. The company guided us in tidying the Fortran code and writing a set of automatic tests that together call all subroutines at least once. After this, Objexx converted the sources automatically, followed by some amount of manual editing. We did a few further manual steps to call the CCP4 libraries from C++, but temporarily still through the Fortran interfaces, to keep the manual changes as limited as possible. The test suite was modified to exercise the new C++ version in addition to the original Fortran. A significant but limited amount of time (approximately a couple weeks) was spent on finding and fixing conversion errors. The majority of errors was due to oversights in the manual changes, which underlines the importance of keeping these at a minimum until all tests are in place.

After the syntax conversion we applied an extensive series of semi-automatic source code transformations. The first major goal was to improve the initially quite poor runtime performance, which was expected. However, it was a surprise when we discovered that the performance of the single-precision exp() function in the GNU/Linux system math libraries is extremely poor compared to the Intel Fortran version which we used for comparison (see the following section in this article). The runtime of several tests is dominated by exp() calls. Therefore we substituted custom code for the exp() function, which is not as fast as the Intel version, but much faster than the math library version. The next important step was to replace all small dynamically allocated arrays with automatic arrays. These changes could be applied globally via a set of small, custom Python scripts. The only localized change was to re-write small functions for reading and writing density maps using low-level C I/O facilities (printf(), scanf()). The re-written functions significantly out-perform the original Fortran subroutines.

The second major goal was to transform the sources to be suitable for building a Python extension. For this, all non-constant global variables had to be converted to dynamically allocated variables, to be allocated when RESOLVE is called from Python, and de-allocated after the call is finished. This was achieved by building a C++ struct holding all Fortran COMMON variables and all function-scope SAVE variables. In essence this means the entire RESOLVE program was converted to one large C++ object which can be constructed and destroyed arbitrarily from Python, in the same process. The encapsulation of the entire program as a regular object was completed by using the C++ stream facilities to either read from the standard input "stdin" or a Python string, and to write all output either to the standard output "stdout" or a Python file object. (To be accurate we mention that some CCP4 library routines write output directly to the standard output, i.e. by-pass the redirections, but for our purposes this is currently not important and can be changed later if necessary.)

After finishing the large-scale source transformations we concentrated on the runtime performance of one of the most important parts of RESOLVE, a search for model fragments. Using profiling tools to find the bottlenecks we were able to manually optimize the code to be faster than the original Fortran code (using the Intel Fortran compiler). This is to illustrate that C++ performance can match Fortran performance if reasonable attention is given to the time-critical parts. Of course, similar manipulations to the Fortran code would make the Fortran speed comparable again. Overall the Fortran version is still faster than the

C++ version, up to ca. factor 1.5-2.0, depending on the platform, but this is mainly due to the use of certain approaches to ease the automatic conversion step. The following section in this article presents a systematic review of runtimes.

We like to emphasize that a fast cycle of making changes, re-compiling, and running the tests was crucial for the timely progress of the conversion work. Luckily, re-compiling and running the tests is easily parallelized. Using a new 24-core system with a very recent Linux operating system (Fedora 11), a complete cycle takes less than five minutes. This would have been unthinkable only a few years ago. Without recent hardware and software, the conversion work would have stretched out much longer and some steps may never have been completed due to developer fatigue.

Systematic review of runtimes

It is difficult to present a meaningful overview of the runtimes of a program as large and diverse as RESOLVE. In our experience each platform and each algorithm has a significant potential for puzzling observations. For this article, we have therefore distilled our observations into a self-contained test case that is small but still sufficiently complex to be representative. The test case is a simplified structure factor calculation which only works for a crystal structure in space group P1, with an orthorhombic unit cell, a constant unit form factor, and isotropic displacement parameters ("B-iso" or "U-iso"). This algorithm was chosen because it is typical for crystallographic applications and it uses the exp() library function (see the previous section).

The Fortran version of the simplified structure factor calculation is:

```
subroutine sf(abcss, n_scatt, xyz, b_iso, n_refl, hkl, f_calc)
implicit none
REAL abcss(3)
integer n_scatt
REAL xyz(3, *)
REAL b_iso(*)
integer n_refl
integer hkl(3, *)
REAL f_calc(2, *)
integer i_refl, i_scatt, j, h
REAL phi, cphi, sphi, dss, ldw, dw, a, b
DO i_refl=1,n_refl
  a = 0
  b = 0
  DO i_scatt=1,n_scatt
    phi = 0
    DO j=1,3
      phi = phi + hkl(j,i_refl) * xyz(j,i_scatt)
    enddo
    phi = phi * 2 * 3.1415926535897931
    call cos_wrapper(cphi, phi)
    call cos_wrapper(sphi, phi - 3.1415926535897931*0.5)
    dss = 0
    DO j=1,3
      h = hkl(j,i_refl)
      dss = dss + h*h * abcss(j)
    enddo
    ldw = -0.25 * dss * b_iso(i_scatt)
    call exp_wrapper(dw, ldw)
    a = a + dw * cphi
    b = b + dw * sphi
  enddo
  f_calc(1, i_refl) = a
  f_calc(2, i_refl) = b
enddo
return
end
```

The corresponding C++ version is:

```
void
sf(real1d& abcss,
   int n_scatt, real2d& xyz, real1d& b_iso,
   int n_refl, int2d& hkl, real2d& f_calc)
{
  int i_refl, i_scatt, j, h;
  float phi, cphi, sphi, dss, ldw, dw, a, b;
  DO1(i_refl, n_refl) {
    a = 0;
    b = 0;
    DO1(i_scatt, n_scatt) {
      phi = 0;
      DO1(j, 3) {
        phi = phi + hkl(j,i_refl) * xyz(j,i_scatt);
      }
      phi = phi * 2 * 3.1415926535897931f;
      cos_wrapper(cphi, phi);
      cos_wrapper(sphi, phi - 3.1415926535897931f*0.5f);
      dss = 0;
      DO1(j, 3) {
        h = hkl(j,i_refl);
        dss = dss + h*h * abcss(j);
      }
      ldw = -0.25f * dss * b_iso(i_scatt);
      exp_wrapper(dw, ldw);
      a = a + dw * cphi;
      b = b + dw * sphi;
    }
    f_calc(1, i_refl) = a;
    f_calc(2, i_refl) = b;
  }
}
```

Obviously this is not idiomatic C++ code and it needs a few lines of support code to follow the Fortran syntax this closely:

```
#define DO1(i,n) for(i=1;i<=n;i++)

template <typename T>
struct dim1
{
  std::vector<T> data;
  dim1(int n) : data(n) {}
  T& operator()(int i) { return data[i-1]; }
};

template <typename T>
struct dim2
{
  int n1;
  std::vector<T> data;
  dim2(int n1_, int n2) : n1(n1_), data(n1*n2) {}
  T& operator()(int i, int j) { return data[i-1+(j-1)*n1]; }
};

typedef dim2<int> int2d;
typedef dim1<float> real1d;
typedef dim2<float> real2d;
```

The `DO1` macro is used to emulate the very concise syntax of Fortran loops. The `dim1` and `dim2` struct templates emulate Fortran column-major arrays (as opposed to row-major C-style arrays) with 1-based indices (as opposed to 0-based C-style indices). The three typedefs lead to slightly more concise code.

In passing we note that a C++ struct is equivalent to a C++ class, but without explicit `private` and `public` keywords all struct attributes are public while all class attributes are private.

For reference, the complete source code is embedded in the file `compcomm/newsletter09/sf_times.py` in the open source cctbx project (Grosse-Kunstleve & Adams, 2003). It includes a driver function for generating a random structure and a random list of Miller indices (two versions, Fortran and C++).

Source codes that are so similar should in theory lead to identical machine code, which means identical runtime performance. However, the limited features of Fortran-77, compared to C++, make it much easier for an optimizer to generate efficient code. To find out how this plays out in practice we ran the test codes above using a selection of Fortran and C++ compilers, with eight different variations of the sources above:

```
"s" or "d": single-precision or double-precision floating-point variables
"E" or "e": using the library exp(arg) function or "max(0.0, 1.0 - arg*arg)"
"C" or "c": using the library cos(arg) function or "arg / (abs(arg)+1.0)"
```

The replacements for the exp() and cos() functions are a simple trick to separate the effects of compile-time optimizations from the runtime performance of the math library. (Obviously, when using the replacement functions the resulting structure factor values are meaningless. To ensure that the algorithm is representative of real calculations if the proper exp() and cos() functions are used, `sf_times.py` includes a numerical comparison with the results of the fully-featured cctbx direct-summation structure-factor calculation.)

The complete results are archived in the file `compcomm/newsletter09/time_tables` in the cctbx project. An example table (one of 22) in the file is:

```
current_platform: Linux-2.6.23.15-137.fc8-x86_64-with-fedora-8-Werewolf
current_node: chevy
build_platform: Linux-2.6.23.15-137.fc8-x86_64-with-fedora-8-Werewolf
build_node: chevy
gcc_static: "-static "
compiler: ifort (IFORT) 11.1 20091012
compiler: GNU Fortran (GCC) 4.1.2 20070925 (Red Hat 4.1.2-33)
compiler: n/a
compiler: icpc (ICC) 11.1 20091012
compiler: g++ (GCC) 4.1.2 20070925 (Red Hat 4.1.2-33)
n_scatt * n_refl: 2000 * 20000

  sEC    seC    sEc    sec    dEC    deC    dEc    dec
  1.67   1.31   0.73   0.46   2.26   1.62   1.25   0.84  ifort
 16.76   5.59  12.97   1.76   7.70   6.35   3.15   1.94  gfortran
 -1.00  -1.00  -1.00  -1.00  -1.00  -1.00  -1.00  -1.00  g77
  1.69   1.33   0.78   0.46   2.20   1.65   1.41   0.84  icpc
 16.66   5.54  12.87   1.69   7.59   6.36   2.95   1.86  g++
```

Each row is for a particular compiler: Intel Fortran (ifort), the current GNU Fortran development line (gfortran), an old GNU Fortran development line (g77), Intel C++ (icpc), and GNU C++ (g++), always in this order, but different versions on different platforms. If a certain compiler is not available on a given platform, the corresponding row is filled with `-1.00`. The table heading (`sEC ... dec`) is not included in the `time_tables` file but was added here do indicate each of the eight source code variations. For

example, `deC` indicates "double precision", using the `max(...)` function instead of the library `exp(...)` function, and using the proper library `cos(...)` function.

The first striking observation is the very large runtime difference between the ifort and gfortran executables. For the `sEC` case the ifort executable is 16.76/1.67 = 10.0 times faster. Comparing the `seC`, `sEc`, and `sec` columns it is evident that the runtime is dominated by the extremely poor performance of the single-precision exp() and cos() library functions. For the `sec` case, which does not call the math library, the gfortran performance compares significantly better, but the ifort executable is still 3.8 faster.

The next surprise is that the double-precision `dEC` gfortran executable is 2.2 times faster than the single-precision `sEC` gfortran executable. From the other double-precision times in the gfortran column it is apparent that the relatively better performance of the double-precision math library functions is the main reasons for the observation. With both math functions replaced the single-precision version is slightly faster than the double-precision version (1.76 vs. 1.94). The performance of the single-precision ifort executables is consistently better than that of the corresponding double-precision ifort executables (factor 1.4 to 1.8).

The timings above show that, for these particular compiler versions, the icpc executables, considering small uncertainties in the times, are as fast as the ifort executables, and the g++ executables are as fast as the gfortran executables. The situation is still very similar for the latest g++ and gfortran versions. The times with gcc 4.4.2 on the same machine ("chevy", 2.9 GHz Xeon) are:

```
sEC    seC    sEc    sec    dEC    deC    dEc    dec
16.39   5.14  12.73   1.63   7.37   6.14   2.92   1.77  gfortran
16.56   5.37  13.02   1.76   7.64   6.34   3.03   1.90  g++
```

Interestingly, a different picture emerges with an older version of the Intel compilers (machine chevy):

```
1.62   1.29   0.73   0.46   2.07   1.65   1.29   0.84  ifort 9.1 20060323
2.67   2.22   1.94   1.50   3.27   2.66   2.62   1.93  icpc 9.1 20061101
sEC    seC    sEc    sec    dEC    deC    dEc    dec
```

The following table compares the Intel C++ times only:

```
sEC    seC    sEc    sec    dEC    deC    dEc    dec
2.67   2.22   1.94   1.50   3.27   2.66   2.62   1.93  icpc 9.1 20061101
1.88   1.43   0.96   0.63   2.44   1.84   1.41   1.02  icpc 10.1 20080312
1.69   1.33   0.78   0.46   2.20   1.65   1.41   0.84  icpc 11.1 20091012
```

Apparently, the Intel C++ optimizer has been improved significantly in recent years, to be on par with the Fortran optimizer.

As an aside, we want to highlight the following result in the time_tables file (machine chevy):

```
sEC    seC    sEc    sec    dEC    deC    dEc    dec
3.20   2.68   1.84   1.41   3.20   2.69   1.84   1.41  ifort 9.1 32-bit
2.06   1.62   1.29   0.84   2.06   1.62   1.29   0.84  ifort 9.1 64-bit
```

Our motivation for showing these times is to encourage the use of 64-bit systems, even on machines with less than 2 GB system memory. We attribute the speed difference to the larger number of general-purpose CPU registers available to a 64-bit application.

We also like to encourage the reader to inspect the `time_tables` file, which contains significantly more information than is highlighted in this article. The current URL to the directory with the file is:

http://cctbx.svn.sourceforge.net/viewvc/cctbx/trunk/compcomm/newsletter09/

Conversion recipe

The RESOLVE conversion process was a major learning experience. From previous smaller conversion projects (FFTPACK and L-BFGS in the cctbx project) we had learned already that a conversion to idiomatic C++ is very time consuming. We also knew that there is very little to no performance gain when converting 1-based Fortran indices to 0-based C-style indices, as is re-confirmed by the results in the previous section. However, we did not have experience encapsulating a complete large program as a Python extension. In the section outlining the RESOLVE conversion work, we already mentioned that the essential idea is to treat the entire program as one large C++ object. Here we present a more detailed recipe that should be applicable to any Fortran-77 program. We do this by way of a small example:

```
subroutine show_resolution(h, k, l)
 implicit none
 integer h, k, l
 real a, b, c
 logical first
 real ass, bss, css
 real dss
 COMMON /abc/ a, b, c
 SAVE first
 SAVE ass, bss, css
 DATA first /.true./
 if (first) then
   first = .false.
   if (a .le. 0 .or. b .le. 0 .or. c .le. 0) then
     write(5, '(1x,a)') 'invalid unit cell constants.'
     stop
   endif
   ass = 1/(a*a)
   bss = 1/(b*b)
   css = 1/(c*c)
 endif
 dss = h*h*ass + k*k*bss + l*l*css
 if (dss .eq. 0) then
   write(6, '(3(1x,i3),1x,a)')
&    h, k, l, '    infinity'
 else
   write(6, '(3(1x,i3),1x,f12.6)')
&    h, k, l, sqrt(1/dss)
 endif
 return
 end

 PROGRAM conv_recipe
 implicit none
 real a, b, c
 COMMON /abc/ a, b, c
 a = 11.0
 b = 12.0
 c = 13.0
 call show_resolution(0, 0, 0)
 call show_resolution(1, 2, 3)
 end
```

This example includes the three major Fortran features breaking encapsulation: COMMON, SAVE, and STOP. COMMON and SAVE introduce global data, which make the subroutine non-reentrant (see http://en.wikipedia.org/wiki/Reentrant_%28subroutine%29). Worse, in this example the reciprocal space parameters (`ass`, `bss`, `css`) are set only the first time the subroutine is called in the process. Clearly, this would not work well for a Python extension. Subsequent calls of a Python function need to be independent of each other to avoid confusing behavior. This is particularly important in large systems.

While it is difficult to pin-point and remove specific critical globals (the program would need to be re-organized), it is not difficult to "globally remove all globals". The first step is to organize the data:

```
struct show_resolution_save
{
  bool first;
  float ass, bss, css;
  show_resolution_save() : first(true) {}
};

struct common
{
  float a, b, c;
  show_resolution_save show_resolution_sve;
  std::ostream & out_stream;
  common(std::ostream& out_stream_) : out_stream(out_stream_) {}
};
```

The previously global data can now be initialized at an arbitrary point in the process, for example:

```
common cmn(std::cout);
```

Now the `cmn` object, holding all data and a reference to the output stream, can be passed to the converted Fortran PROGRAM which is simply a normal function in C++:

```
void
conv_recipe(common& cmn)
{
  cmn.a = 11.0;
  cmn.b = 12.0;
  cmn.c = 13.0;
  show_resolution(cmn, 0, 0, 0);
  show_resolution(cmn, 1, 2, 3);
}
```

This function assigns values to three `cmn` data members and then passes the `cmn` object on to the show_resolution() function:

```
void
show_resolution(common& cmn, int h, int k, int l)
{
  show_resolution_save& sve = cmn.show_resolution_sve;
  if (sve.first) {
    sve.first = false;
    if (cmn.a <= 0 || cmn.b <= 0 || cmn.c <= 0) {
      throw std::runtime_error(
        "invalid unit cell constants.");
    }
    sve.ass = 1/(cmn.a*cmn.a);
    sve.bss = 1/(cmn.b*cmn.b);
    sve.css = 1/(cmn.c*cmn.c);
  }
  float dss = h*h*sve.ass + k*k*sve.bss + l*l*sve.css;
  std::ostream& cout = cmn.out_stream;
  if (dss == 0) {
    cout << boost::format(" %3d %3d %3d     infinity\n")
      % h % k % l;
  }
  else {
    cout << boost::format(" %3d %3d %3d %12.6f\n")
      % h % k % l % std::sqrt(1/dss);
  }
}
```

This implementation is slightly more verbose than the original Fortran version, but it is also more readable because it is immediately clear which data are related to the `cmn` and `sve` objects. We note that it is possible to avoid the `sve`. if desired by converting `show_resolution()` to a member function of the show_resolution_save struct. In RESOLVE we preferred the approach shown for clarity.

The `cmn` object can be instantiated from a C++ main or a Python function. Currently, RESOLVE makes use of both possibilities. In the case of the Python function, as soon as the call is finished the `cmn` object goes out of scope and the space for all its data is released.

The complete `conv_recipe.cpp` can be found under the URL shown at the end of the previous section. It can be compiled on virtually any Linux system with a development environment since the boost library, which is used for formatting the output of the example, is usually included in the standard development environments.

We think following the conversion recipe shown above it is feasible to automatically convert large sections of entire Fortran programs to re-usable extensions of large integrated systems such as PHENIX. The main obstacle is the Fortran I/O system, which is not trivial to emulate. The seasoned "f2c" system (http://netlib.org/f2c/) proves that it is possible with a reasonable effort, but in practice it may be wisest to simplify the Fortran I/O first and to apply program-specific custom scripts combined with a small amount of manual changes to solve problems with complex I/O operations.

Conclusion

Based on our experience working on CNS (Brunger et al. 1998), RESOLVE, and PHENIX, we are convinced that writing an entire program system in Fortran or even C++ is a very inefficient use of the most valuable resource, human time. Within the PHENIX project, most new developments start out as Python scripts building on previously implemented methods. By converting RESOLVE to a Python extension we have added the RESOLVE functionality to the pool of methods that can easily be re-used. Our systematic review of runtimes suggests that this can be achieved without significantly compromising the performance of time-critical algorithms.

The conservative conversion methods presented above do not produce a designed "clean" system. However, they lead to a system that works today and is simultaneously open to evolutionary development. If there is an evolutionary path there is no need to replace the old. It can be modified instead. In the particular case of RESOLVE, we anticipate gradual changes that over time lead to an increasingly modular organization. Post conversion, the RESOLVE code has already been changed to make use of the extensive C++ libraries developed previously in the context of PHENIX (notably the symmetry and the fast Fourier transform libraries). We also anticipate that some RESOLVE algorithms can be re-factored for use in other contexts, to accelerate the development of PHENIX as a whole.

Acknowledgments

We gratefully acknowledge the financial support of NIH/NIGMS under grant number P01GM063210. Our work was supported in part by the US Department of Energy under Contract No. DE-AC02-05CH11231. We like to thank Stuart Mentzer (Objexx Engineering, Inc.) for invaluable help with the bulk conversion of RESOLVE. Luc Bourhis' code for the C++ stream redirection (part of the cctbx project) was critical for fully encapsulating RESOLVE.

References

Abrahams, D., Grosse-Kunstleve, R.W. (2003). C/C++ Users Journal, 21(7), 29-36.

Adams P.D., Grosse-Kunstleve, R.W., Hung, L.-W., Ioerger, T.R., McCoy, A.J., Moriarty, N.W., Read, R.J., Sacchettini, J.C., Sauter N.K., Terwilliger, T.C. (2002). Acta Cryst. D58, 1948-1954.

Brunger, A.T., Adams, P.D., Clore, G.M., DeLano, W.L., Gros, P., Grosse-Kunstleve, R.W., Jiang, J.-S., Kuszewski, J., Nilges, M., Pannu, N.S., Read, R.J., Rice, L.M., Simonson, T., Warren, G.L. (1998). Acta Cryst. D54, 905-921.

Collaborative Computational Project, Number 4 (1994). Acta Cryst. D50, 760-763.

Grosse-Kunstleve, R.W., Adams, P.D. (2003). Newsletter of the IUCr Commission on Crystallographic Computing, 1, 28-38.

Terwilliger, T.C. (2000). Acta Cryst. D56, 965-972.

Terwilliger, T.C. (2003). Acta Cryst. D59, 38-44.

Crystallographic Computing in Glasgow: The GX Program System

Kenneth W. Muir
Chemistry Department, University of Glasgow, Glasgow, G12 8QQ, United Kingdom. E-mail: ken@chem.gla.ac.uk

Introduction

The GX crystallographic program system for 32-bit minicomputers (Mallinson & Muir, 1985) was used for about a decade to convert the output of the Nonius CAD4 diffractometers in Glasgow into completed diffraction analyses. For our lab it represented a transitional stage in crystallographic computing: before GX most of our work was done by batch calculation on mainframes; GX became obsolete when personal computers could handle the calculations.

GX was designed so that any program could easily be incorporated into the suite if it adhered to a simple set of protocols. Looking back nearly thirty years later, I feel the effort we put into GX was worthwhile. Although its successor, the much better-known and widely-used WinGX package devised by my colleague Louis Farrugia (Farrugia, 1999), contains little of the original GX code, it retains the idea that programs by different authors should talk to one another, thereby widening access to the best free software.

In order to see where GX came from, a short history lesson is appropriate

The Early History of Crystallographic Computing in Glasgow

The crystallography group in Glasgow was set up in 1943 by J.M. Robertson. Earlier, he had shown that the structures of large organic molecules could be determined by diffraction methods (Robertson, 1935). Up to his retiral in 1970 his group was very successful in solving complex organic structures, often by working on heavy atom derivatives; his successor, George Sim, widened the group's interests to include organometallics and even proteins.

A snapshot of crystallographic computing about 1960 is provided by the proceedings of a conference held in Glasgow in 1959 (Pepinsky, Robertson & Speakman, 1961). A later book edited by John Rollett (1965) is also still interesting; the chapters on crystallographic mathematics written by Rollett himself have an elegance not evident in other textbooks of the period.

When I joined the Glasgow group in 1963, it was still using the fairly primitive DEUCE computer which took about ten hours to do an electron density synthesis. Its replacement by an English Electric KDF9 was therefore eagerly expected. It was hoped that the new machine would increase computing speeds by about two orders of magnitude. A compiler for a high-level programming language, ALGOL, would also greatly simplify writing programs. However, the most important of the arrivals in Glasgow at this time was Durward Cruickshank, who came to take up the newly-created Joseph Black Chair. One of his projects was to write an ALGOL crystallographic least squares program with J.G.F. Smith. Cruickshank tried very hard to get all eight U.K. crystallography laboratories with access to KDF9s to collaborate in writing software and very kindly invited Douglas Macgregor and myself, very junior research students with interests in programming, to attend his meetings. Despite Cruickshank's best efforts, this inter-university collaboration produced little in the way of communal software.

In the event, the Glasgow group relied mainly on its own ALGOL software based round the Cruickshank & Smith least squares program and a Fourier program written by Jamie Sime. Douglas Macgregor and I developed the over-ambitiously named ASS (Automatic Structure Solution) package. This allowed us to do our structures efficiently. However, we had plenty of time to play with new ideas for programs during

KDF9's first year; program development was possible but not serious production computing – a development compiler came with KDF9 but another year went by before the main Kidsgrove ALGOL compiler arrived.

We wanted to transfer to the computer some of the tedious manual tasks involved in crystal structure analysis. We experimented with some primitive routines for automatic map interpretation and model refinement. Thus, we wrote a program which searched Fourier maps for peaks (and/or holes). This was a great improvement on the previous system used in the lab. One first wrote by hand the electron density function values in each 2D section onto tracing paper, contoured the section and then transferred the contours onto a two-foot square glass sheet. These sheets were then stacked for interpretation of the 3D map. Later we discovered that John Rollett had also written a peak search routine at pretty much the same time. The preparation of typed coordinate tables for publication was another time-consuming and error-prone chore; accordingly, the ASS system provided for automatic generation of such tables for publication on a paper-tape-fed electric typewriter. Many colleagues made contributions to ASS and some idea of what it could do in its final developed form can be obtained from the index page of the complete listing of the ALGOL code by Pollard (1968) - see Table 1.

Table 1. The ASS Program List .

Some methods tried in the 1960s have not survived into modern crystallography. The *Minimum Residual Method* of Bhuiya and Stanley (1963) was quite good at refining projection data where data/parameter ratios are dangerously low for conventional least squares. It was briefly fashionable in Glasgow in the mid-1960s. The *linear* least squares determination of *y*-coordinates, given a solved *h*0*l* projection, shows the fairly desperate lengths to which one might go before direct methods became widely available (Muir & Robertson, 1972).

In the late sixties I spent a couple of years as a post-doc with Jim Ibers at Northwestern while my wife worked with Walter Hamilton at Brookhaven. This was a formative experience. They were both skilled crystallographic programmers with a no-nonsense approach and a real passion for getting things right. I was also impressed by the ability of American scientists to work collaboratively when it made sense to do so.

About 1970 KDF9 was decommissioned. Crystallographic computing in Glasgow then moved to FORTAN-based systems running on mainframes, first to Stewart's XRAY system and then to SHELX-76. Batch computing on a mainframe had its frustrations. There might be a long wait to find that a job had failed for some trivial reason. Finding the optimum view direction for an ORTEP plot could take a week or two.

The GX System

By 1980 it was becoming apparent that a dedicated 32-bit-minicomputer could offer the crystallographer very superior facilities compared with a mainframe and at an acceptable cost. We therefore decided to develop a suite of programs suitable for an environment in which the crystallographer sat at a terminal and did most of his (or her) calculations in real time. The availability of a single- or multi-user operating system with file management and editing facilities and a FORTRAN 77 compiler was assumed. The minimum hardware requirement for our system was set at 0.5 Mbyte program address space, 10 Mbyte disc store, a keyboard terminal with graphics capability, printer and pen plotter (or an equivalent hard copy graphics device). To put this hardware requirement into perspective, I am writing this article on a desktop PC which has a 2 Gbyte address space and 500 Gbytes of disk storage, respectively 4000 and 50,000 times the minimum values specified for a machine running GX. The PC does crystallographic least squares refinement about 1000 times faster than the SEL 32/27 on which GX was developed; a calculation which today takes a few seconds would have required about an hour on the SEL32/27.

Table 2 lists the programs available in GX. The GX manual (see Appendix) gives some idea of how the system looked to the user. Compared with ASS the trend is to a smaller number of programs, each doing rather more tasks. Note that it was still worth saving refinement time by using a separate block diagonal least squares program

Table 2 Programs in the GX package

ABSORB	Gaussian quadrature absorption correction
BLOCK	Block diagonal least squares refinement
DIFABS	Walker & Stuart empirical absorption correction
FFT	FFT Fourier program (by Ten Eyck, taken from Main's MULTAN package)
FTAB	Structure factor tables
GEOM	Bond lengths, angles, contacts, planes etc.
ORTEP	Interactive version of Johnson's ORTEP
RBLS	Full matrix least squares refinement with constraints and rigid groups
SEARCH	Peak search
SORT	Sort and average equivalent intensities
WTANAL	Weighting analysis
XYZ	Operations on the model file

Notes
(a) Data reduction and direct method calculations were available outside GX.
(b) All these programs used the GX subroutine library described in the text.

Some of the key features of GX are worth describing briefly.

The Binary Data File

For each Bragg reflection the batch number, h, k, l, F^2, $\sigma(F^2)$, path length (for extinction corrections) and reflection sequence number (a measure of X-ray exposure) were packed into four 32-bit words. This file was kept on the disk during active computation. Packing was necessary to economise on disk space which was expensive and therefore scarce. Even after we added a 64-Mbyte auxiliary disk at a cost of £15,000 - quite a lot of money in 1986 - disk storage had to be carefully managed.

The Model File

For each structure there was at least one editable text file, the model file, which described the structure. All programs used the subroutines MDLIN(IN) and MDLOUT(IOUT) to read this file from channel IN and write it to channel IOUT. MDLIN set up the common blocks MODELC and MODEL.

```
      CHARACTER COMPID(72),LINE(148)
      CHARACTER*8 LABEL(200),FTYPE(13)
      INTEGER CN(200),CONTNT(13),FCH(20),FWD(20)
      COMMON /MODELC/COMPID,LABEL,FTYPE
      COMMON /MODEL/ NATOM,NR,NT,NTYPE,NFTYPE(200),P(3,200),
     1CN,SOF(200),UIJ(6,200),EP(3,200),ESOF(200),EUIJ(6,200),
     2WAVEL,CELL(6),ECELL(6),R(24,3,4),T(3,4),ICENT,
     3SFAC(14,13),CONTNT,SCALE(4),ETA,NQ,EXTNCT
```

COMPID is a title array, LABEL contains up to 200 eight-byte atom labels, FTYPE the scattering factor labels. The model has NATOM atoms, NR equivalent positions, NTYPE scattering factors. For each atom NFTYPE, P, CN etc. contain the scattering factor type, coordinates, coordination number and so on. Some typical model file entries are shown below (some necessary SYMM entries are omitted).

```
TITLE TEST CASE P6122 ACTA CRYST., 1972, A28,384.                        0001.000
CELL      8.530     8.530    20.370   -.00000   -.00000   -.50000       0002.000
LATT  A  P                                                               0003.000
SYMM  - Y, + X - Y, 1/3 + Z                                              0004.000
SYMM  - X + Y, - X, 2/3 + Z                                              0005.000

SFAC S         6.90530  1.46790  5.20340 22.21510  1.43790   .25360 =   0015.000
   1.58630  56.17200    .86690    .350    .869   512.500   1.100   1.9000016.000
SFAC O         3.04850 13.27710  2.28680  5.70110  1.54630   .32390 =   0017.000
    .86700  32.90891    .25080    .000    .000    30.480    .850   1.4000018.000
SFAC C         2.26069 22.69070  1.56165   .65667  1.05075  9.75618 =   0019.000
    .83926  55.59489    .28698    .000    .000    10.670    .850   2.0000020.000
CONTENTS 6S 6O 12C                                                       0021.000
ATOM S           .20200    .79800    .91667    .50000 CN0 =              0022.000
    .02533    .02533    .02533    .01267    .00000    .00000            0023.000
ATOM O           .49800    .49800    .66667    .50000 CN0 =              0024.000
    .02533    .02533    .02533    .01267    .00000    .00000            0025.000
ATOM C(1)        .48800    .09600    .03800   1.00000 CN0 =              0026.000
    .03166    .00000    .00000    .00000    .00000    .00000            0027.000
```

The content of each line is defined by a initial key word and other entries are in free format. Subroutine PARSE was used to divide each line into fields. Each field was then converted as appropriate into a string variable, a binary floating point variable or an integer variable by the routines AFORMT, FFORMT and IFORMT. A FORTRAN subroutine copied from an existing ALGOL procedure was used to convert equivalent positions into R and t matrices. Likewise, atom names followed the International Union conventions embodied in the Cruickshank & Smith Least Squares Program e.g. C(1) – a conventional chemical element symbol such as C, usually referring to an internal database of scattering factors, plus an optional sequence number in parentheses.

These library subroutines for reading files made it straightforward to write a new program or to convert an existing program so that it could use the standard structure files.

Program Instructions

A break with mainframe program conventions was that instructions to the program were regarded as logically separate from the structure files. A big effort was made to present on-screen menus to the user to avoid frequent recourse to printed manuals. For example the program XYZ, a rag-bag of simple operations on the model file offered the following on-screen options, some of which prompted for further input.

```
   MENU OF OPERATIONS ON THE MODEL FILE
1. INVERT    - Invert configuration of model
2. TIDY      - Sort atoms into logical order
3. RING <D>  - Construct planar polygon of side D,
               D = 1.395 A if not specified by user
4. CON R <T> - New x = Rx+T; type R in rows then T. T is optional.
5. CENTROID NAME ATOM1 ATOM2... - Include centroid in model file
6. RESID <ATOM1 ATOM2 ...> - Collect atoms into connected residues.
              ATOM1 defaults to the first atom in the model file.
              Otherwise give one atom per residue
7. FIXEL EL1 EL2 ..- Reset U = 0.05 by element
8. FIXAT ATOM1 ATOM2 . - Reset to Uiso = Ueq by atom
9. FIXH  <DIST> <UFAC> - Tidy H-atoms. DIST = X-H and U(H) = UFAC * Ueq(X).
             DIST = 0.96 A, UFAC = 1.2 by default
```

As one would expect, the full matrix least squares refinement program RBLS used much the largest amount of machine time. RBLS could be run interactively but for longer calculations it made more sense to type refinement instructions into a file and then quickly check that that the calculation would run correctly. The checked calculation could then be left in a batch queue to run to completion overnight.

Coordinate Covariances

One minor elaboration in GX proved useful a decade later. The distance and angle program GEOM could get bond length standard uncertainties (s.u.s) either from the RBLS normal matrix (the preferred option) or approximate them from the coordinate s.u.s. In the latter case allowance was made for covariance between fractional coordinates in oblique coordinate systems using the cell angles, as suggested by Templeton (1959).

In the 1990s I was working as a co-editor of *Acta Crystallographica*. The staff in Chester used to check each submitted CIF, highlighting to the co-editor any discrepancies between their results and those provided by the author. The advent of SHELXL-93, which used the full normal matrix in error calculations, brought a sudden increase in reported discrepancies. Eventually, it turned out (Mallinson & Muir, 1993) that the discrepancies arose because the checking programs, unlike GEOM, did not allow for covariance between fractional coordinates in oblique coordinate systems. Prior to general use of SHELXL-93, the *Acta* checking process found no discrepancies because the programs used by submitting authors and those used in Chester both made the same invalid approximation. The moral: always remember that a satisfactory pass through a checking system is no guarantee that everything is fine. The checkers and the checked may both have made the same mistake.

Concluding Remarks

Modern crystallographic software is vastly more sophisticated than the crude routines which made up ASS and the rather more refined ones embodied in GX. The changes have been mainly of three kinds. (1) Programs today are usually designed to work for all space groups. In the 1960s you could not even be sure that a program would correctly handle triclinic axes. This is understandable if one realises that a doctoral thesis might then easily contain only four or five structures, none triclinic. (2) More tasks are handled automatically, setting the parameter constraints demanded by special positions being an obvious example. (3) Program design assumes, often tacitly, a particular operating environment and type of hardware. As I have tried to show, these have both changed dramatically in the last forty years.

Acknowledgements

Computer programs typically embody the work and ideas of many individuals. It is therefore difficult, in an article of this kind, to note every significant contribution. Speaking personally, I would like especially to acknowledge those of Ronnie Pollard, Walter Macdonald and Douglas Macgregor to ASS and that of Paul Mallinson to GX. I have also learned much, directly and indirectly, from an earlier generation of crystallographic programmers, notably Durward Cruickshank, John Rollett, Jim Ibers and Walter Hamilton. More recently, I have enjoyed many stimulating crystallographic discussions with Louis Farrugia.

References

Bhuiya, A.K. & Stanley, E. (1963). *Acta Cryst*., **16**, 981 – 984.
Farrugia, L.J. (1999). *J. Appl. Cryst*., **32**, 837 – 838.
Mallinson, P.R. & Muir, K.W. (1985). *J. Appl. Cryst*., **18**, 51 – 53.
Mallinson, P.R. & Muir, K.W. (1993). *J. Appl. Cryst*., **26**, 142 – 143.
Muir, K.W. & Robertson, J.M. (1972). *Acta Cryst*., **B28**, 879 – 884.
Pepinsky, R., Robertson, J.M., & Speakman, J.C. (1961). *Computing Methods and the Phase Problem in X-ray Crystal Analysis*, Pergamon Press, Oxford, U.K.
Pollard, R. (1968). The ASS Program System. University of Glasgow. U.K.
Robertson, J.M. (1935). *J. Chem.* Soc., 615 – 621.
Rollett, J.S. (1965). *Computing Methods in Crystallography*, Pergamon Press, Oxford, U.K.
Templeton, D.H. (1959). *Acta Cryst*., **12**, 771 – 773.

Appendix The GX Program Manual

FORTRAN code for the GX system as described above can be obtained via:

http://www.chem.gla.ac.uk/~louis/software/

Editor's note: the GX manual is listed starting on page 394 within Addendum B: Computing Software manuals and reference materials. The GX source code is also archived as a zip file with this newsletter.

The Quick and Dirty Crystallographic Computer Program

Joseph H. Reibenspies
Department of Chemistry, Texas A & M University, College Station, Texas, 77842, USA. E-mail: j-reibenspies@tamu.edu

Here is an old puzzle. If a bottle and a cork cost $1.10 and the bottle costs $1.00 more than the cork, what is the price of the cork? If you said a dime ($0.10) you would not be alone, and you would not be correct. Here is the math: [1.10=x+(1.00+x) : x= 0.05]. The cork cost 5¢ and the bottle costs $1.05. Do not worry; it took a pencil and paper for me to figure this one out. The real question is why everyone automatically thinks the cork costs 10¢.

People do not think like computers. They do not solve problems in the same manner, so when the cork and bottle puzzle is presented to a normal person, they solve it employing the methods that they were taught or learned over a lifetime of experiences. A computer would solve the puzzle as if it were a mathematical problem. The computer can perform computations faster and more reliably than any normal human, but when it comes to problem solving that is a different story. Humans have evolved the skill to solve problems through trial and error. The computer "learns" through upgrades.

The interface between the human and the computer is the program. The computer program can be as simple as the old first code "Print Hello", to as complicated as a weather prediction suite. No matter what form the program takes, it represents the grey area where human problem solving collides with computer bit shuffling. The evolution of program coding is a gradual thing. The programmer does not begin a project by typing in millions of lines of code, compiling the beast and hope for the best. All large sophisticated programs began as small bits of code that accomplished a particular task. These bits of code, sometimes referred to as a "quick and dirty program" were then gathered together into large suites of programs. Certain bits of codes that were useful to many different tasks were gathered together to form object libraries. These libraries were employed again and again like bricks in a wall to form larger more complicated programs. Finally these programs themselves were embedded into yet other master programs to form even larger program packages that were even more advanced. This is the evolution of the modern crystallographic program, from its basic small bit codes to the advanced crystallographic program systems.

The higher you go up the computer program evolutionary ladder the easier it is to employ human problem solving techniques to crystallographic puzzles. The user is separated from the task of how to sum the Fourier, which leaves them only the question of why or when to perform the sum. The small bit of code, written in many cases by an unacknowledged and forgotten programmer, is the interface between human thought and computer brawn. The average crystallographic user no longer needs to hammer code to invert a matrix or format a table. These tasks fall to professional programmers who often use the object libraries written years before their time to write their code. It is almost reminiscent of a researcher combing a library for a forgotten book to add to their thesis; however in the case of the programmer, there is no reference and plagiarism is the accepted fact.

This brings us back to the quick and dirty crystallographic program. There will always be new ideas that will need to be interfaced to a computer. These ideas will need to be formed into algorithms which in turn will need to be coded into programs. Somewhere someone will take their, or someone else's, crystallographic thoughts to the next level. There are always problems to be solved and there will always be someone willing to invest the time to "think" like a computer to solve them. They may achieve recognition for their work, but more likely than not, they will join the anonymous programmers that have preceded them.

There is help for the occasional programmer. References books such as the "Numerical Recipes" series can keep the programmer from re-inventing the wheel so to speak. There is no need to code a least

squares program from scratch, when one can find better code on-line or in a book. The occasional programmer should keep themselves close to their problem. There are still many crystallographic problems to be solved. Let the professional programmers solve the graphical user interface questions; the crystallographer should use their education to tackle problems that suit their skills.

There is also help from the old masters. Those who have preceded us have devoted significant time and effort to solve basic crystallographic programming problems. The on-line resources such as CCP-14 http://www.ccp14.ac.uk/ and CCP-4 http://www.ccp4.ac.uk/ catalog many programs that you may find useful and Armel Le Bail has collected "old" code in an on-line museum http://sdpd.univ-lemans.fr/museum/, which contains so called legacy programs that will never be out-dated.

The quick and dirty program is continually evolving and finding new canvases to paint. It has come a long way from the early machine code and Fortran programs to Web based Javascripts and distributed computing. Whatever its form the crystallographer will adapt and perhaps find new resources to gap the expanse of human thought and computer logic.

So what was the price of the cork? For most users they are content to wait for someone to write an applet for their iphone to solve the problem, but a few will take up the challenge and solve the problem themselves and code that applet. Let us all hope that the challenge of solving the problem will always remain worth the time invested.

1974-76: The first version of CAOS for the HP 21MX minicomputer

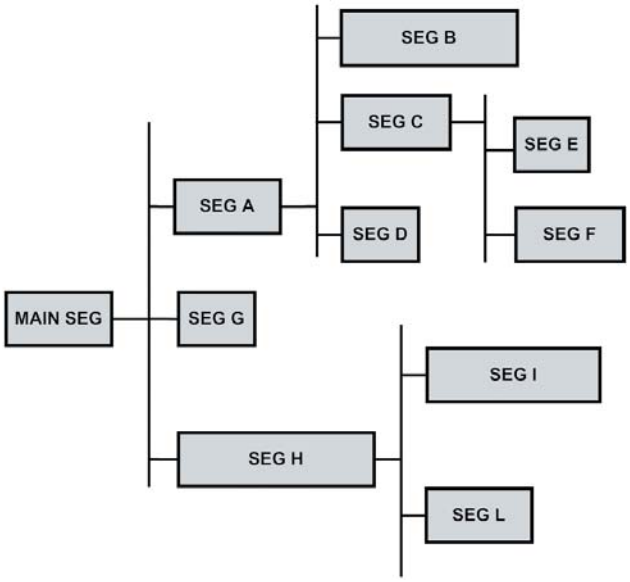
Riccardo Spagna
Istituto di Cristallografia - CNR, Sede di Monterotondo, Area della Ricerca Roma 1, Via Salaria Km.29, 00016 Monterotondo Stazione (Roma), Italy. E-mail: riccardo.spagna@ic.cnr.it

Until the early 1970's, the Institute of Structural Chemistry (ISC) (now Institute of Crystallography) was located in Rome University "La Sapienza" and used the central computer of the University, UNIVAC 1108 (later 1110), computer of great power for the standards of the time. This mainframe had a RAM of 65K (later 128K) 36-bit word (almost all computer manufacturers of that time delivered 36-bit systems with 6-bit character including IBM, DEC) and, to run large applications in this limited memory environment, UNIVAC applications had to be broken up into segments.

Image of a UNIVAC 1108 from http://en.wikipedia.org/wiki/UNIVAC_1108

A segmented program produced by MAP (the linking system) consisted of one main segment (which resided in main storage throughout the execution of the program), and subordinate segments (which were loaded into main storage as they were needed).

Therefore, UNIVAC program was similar to a tree, whereas only one branch was in memory at a time and the elements assigned to that branch didn't interact with the elements of another branch, and the main segment formed the trunk of the tree.



When the Exec program loader executed a segmented program, the main segment was loaded first and the subordinate segments were loaded whenever they were referenced. If the code needed an element stored in an unloaded segment, it had to call a routine that resided in the main segment and another call loaded the referenced segment into memory. Thus, the system "paged" (swapped) whole banks into and out of memory.

Based on these characteristics, a crystallographic program was written, SYSTEM (Carruthers & Spagna, 1975), which allowed the processing of a series of calculations, from the structural determination through the Patterson map, to the refinement of the atomic parameters, then to the geometric calculations. This program was created based on information gathered from former crystallographic programs (Domenicano *et al*., 1969).

In 1973, the ISC moved from the University to the Research Area that the Italian National Research Council was building in the countryside, north of Rome, where the closest town is about 10 km away. To keep using the UNIVAC 1110, a HP 2100 minicomputer (which had just appeared on the market) was purchased.

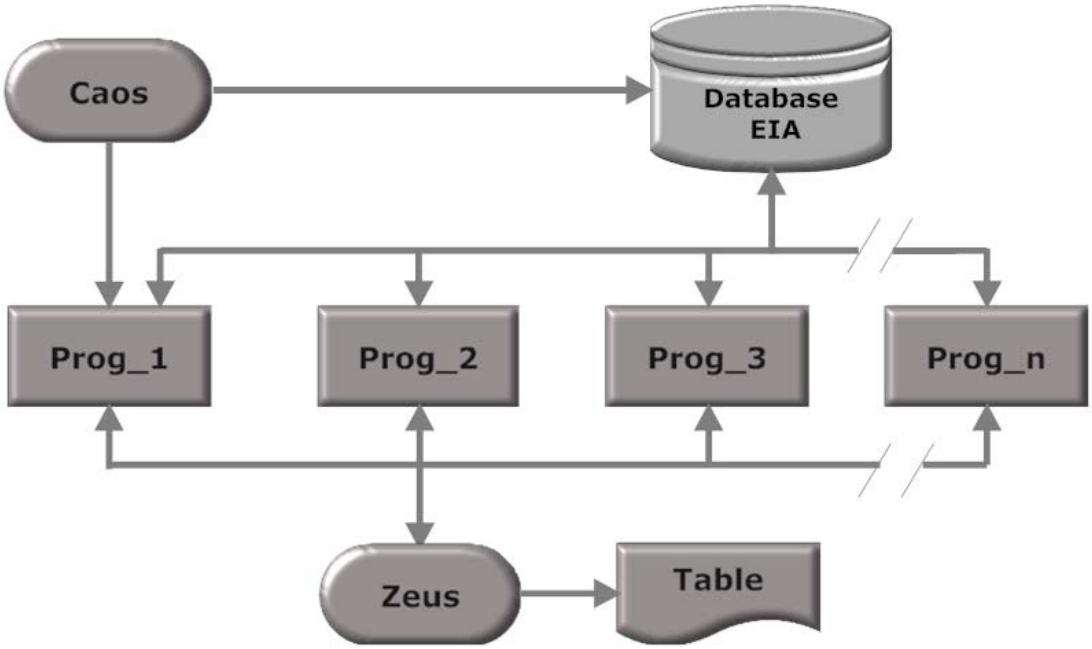
After establishing a radio link connection to the mainframe of the University, the HP terminals became points of remote access. This minicomputer had a memory capacity of 32K, 16-bit word, it was controlled by a Disc Operating System (DOS) and it could run a good Fortran compiler. Quickly we recognized the great potential it had to offer, particularly in the later version, HP21MX, which ran the HP RTE (Real Time) Operating System (Cerrini *et al*., 1975). Other users also started looking at these minicomputers as valid alternatives to the large mainframes(Shiono, 1970). In fact, to consider the “total” time necessary for an elaboration the user in a remote center has to take into account the time employed dispatching the batch with the data for the calculation and the necessary time to get back the answer. The time of the elaboration is only a small fraction of the turnaround time. When using a local minicomputer, the calculation time of the CPU is much longer as compared to that of using a mainframe; however, the results appear much faster.

Image of a HP2100 from http://ed-thelen.org/comp-hist/hp-2100-system.jpg

Therefore, a new version of the crystallographic program SYSTEM was created based on the knowledge gathered in Oxford with the collaboration of J.R. Carruthers on the creation of the crystallographic system CRYSTALS (Carruthers & Rollett, 1975; Betteridge, *et al*., 2003).
Although RTE had a loader that allowed the segmentation of the program, the very small RAM available caused us to return to a group of single programs. The new system of crystallographic programs was called CAOS (Crystallographic Analysis Operating System) (Cerrini & Spagna, 1977). During the creation of CAOS, the following factors were taken into account:

1) none of the programs was allowed to occupy more than 32K, 16-bit word of core memory;
2) simple introduction of new programs to perform new calculations, without having to modify the current parts;
3) the user had to continue to "see" a unique program as on the UNIVAC;
4) choice of user to run the program in batch or interactive mode.

Therefore, program system CAOS was organized into two small programs, Caos and Zeus, and in some programs Prog_1, Prog_2, etc…, Prog_n, devoted to crystallographic calculations. (See flow chart below)



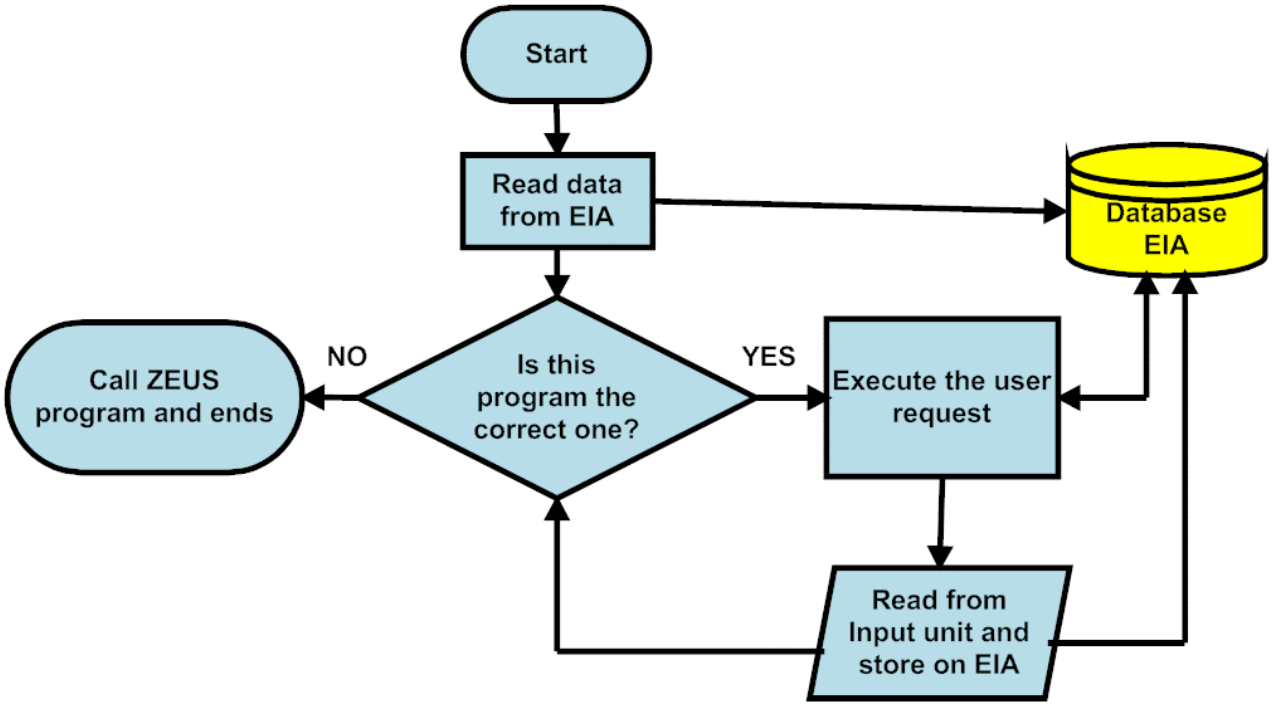
To run CAOS in interactive mode, we followed the approach taken in the SYSTEM method and improved in the CRYSTALS program, by creating a database of the crystallographic information for each crystal structure analysis on a direct access file. Following Rollett's suggestion (Rollett, 1970), the data was organized as "lists" and each list grouped together related information (Carruthers, 1977). In the initial run, the user gave the basic crystallographic data to the program, i.e.; cell parameters, space groups, content and reflections, and the program then stored them as lists on the database. Therefore, in subsequent runs, the user only had to input very few commands to compute the crystallographic calculations. This method remained unchanged in the subsequent versions of CAOS. Eventually, it was used in new programs of structural determination with the direct methods and refinement SIR (Burla, *et al*., 2005) and ultimately in the more complex program "IlMilione" (Burla, *et al*., 2007).

To guarantee the exchange of information among the single programs of CAOS, we decided to reserve the initial space of the direct access file, where the database of crystal structure was established. The length of this space was fixed in 1,600 words and was named the EIA (Exchange Information Area).
Like the SYSTEM and CRYSTALS programs, the user gave commands (or directives) to CAOS, which enabled the program to perform. Each command was a string of characters preceded by a "#" hash symbol.

For example, the directive "#fourier" was given to compute a map of the electronic density reading the lists of interest (cell parameters, structure factors, and so on) from the database of the crystal structure. Because RU was an implied command of the RTE operating system, CAOS started just by entering the string "Caos". The following tasks were performed by the program:

1) obtain the name of the direct access file for the structure and the I/O physical units
2) read the first directive and then the subsequent records from the input unit until the next directive (with '#" in first position) is found or entered
3) write all the input records in the EIA space
4) execute Prog_1 through the proper call EXEC, passing the name of the structure file, and ended.

Prog_1 read the first directive from the EIA and checked if it could fulfill the request. If yes, it executed the crystallographic calculation, updated the structure file, read the next directive from the EIA and the subsequent records from the input file until the "#"character, shifted this block of records in first position and checked if it could satisfy the request. If yes, the loop would continue. If not, Prog_1 executed the switch program Zeus and ended. (See diagram below)



The program Zeus read the first directive from the EIA and a table from formatted file containing all the available directives and which program could execute. Then, Zeus executed the proper Prog_n program and ended.

The Prog_n worked like Prog_1 and the run ended with the directive "#end".

To maintain the size of the single program within the 32K, 16-bit word, we had to consider the dimension of the source code, the data storage and, of course, the O.S. Where it was possible, a crystallographic calculation (fourier, refinement, distance and angles, etc…) was done by a single program. Consequently, the dimension of the source code depended on the complexity of the problem to compute, the ability of the programmer and the level of optimization applied. Conversely, the dimension of the data storage depended on the number of elements in the data arrays that we had to declare *a priori* at the beginning of the program. For example, we could decide that the maximum number of atoms handled by CAOS was 100 and then we had to declare in FORTRAN language:

DIMENSION X(100),Y(100),Z(100) The dimension of the arrays for the positional atomic parameters *x,y,z*. If we have a compound with 35 atoms, we waste memory space and on the contrary, if we have 101 atoms, we have to modify the dimension and recompile the program.

To reduce the dimension of each program without *a priori* limitation of the maximum number of atoms and reflections, we decided to avoid the use of specific arrays to store the crystallographic data, but declared only one large array in a COMMON block:

COMMON/DATA/ STORE(10000) and a set of pointers to access the data. This approach was used by the program SYSTEM and was also followed by the program CRYSTALS. For example, we declared in the following COMMON statement four variables for the atomic parameters:

COMMON/LIST5/ N5,M5,L5,MD5 where N5 was the number of atoms of the structure, L5 was the start address in the vector STORE of the atomic parameters, MD5 was the number of parameters or each atom and M5 is the current address of the atomic parameters. The following loop explains the technique:

```
        M5=L5
        DO 1000  I=1, N5
        X=STORE(M5)
        Y=STORE(M5+1)
        Z=STORE(M5+2)
        ……...
        (fortran statements using the variables X,Y,Z)
        ……..
        M5=M5+MD5
1000    CONTINUE
```

We used this procedure for all the variables in which the number of depends on the compounds under analysis, easily controlling the size of the data storage.

References

Betteridge, P. W., Carruthers, J. R., Cooper, R. I., Prout, K., Watkin, D. J. (2003). J. Appl. Cryst. 36, 1487.

Burla, M.C., Caliandro, R., Camalli, M., Carrozzini, B., Cascarano, G.L., De Caro, L., Giacovazzo, C.,Polidori, G., Spagna, R. (2005), *J. Appl. Cryst.* **38**, 381-388.

Burla M.C., Caliandro R., Camalli M., Carrozzini B., Cascarano G.L., De Caro L., Giacovazzo C., Polidori G., Siliqi D., Spagna R. (2007), *J. Appl. Cryst.* **40**, 609-613.

Carruthers, J.R., Rollett, J.S. (1975) Manual of CRYSTALS system on the ICL 1900

Carruthers, J.R., Spagna R. (1975) VII AIC Meeting, Abtract pag 65.

Carruthers, J.R. (1977) IV ECM, Abstract Ob. 2

Cerrini S., Colapietro M., Spagna R. (1975) VII AIC Meeting, Abstract pag 68

Cerrini S., Spagna R. (1977) IV ECM, Abstract A212

Domenicano A., Spagna R., Vaciago A. (1969) *Rend. Accad. Lincei*, **47**, 331.

Rollett, J.S. (1970) *Crystallographic Computing*, Edited by Ahmed F.R., Munksgaard, Copenhagen, p. 302

Shiono, R. (1970) *Crystallographic Computing*, Edited by Ahmed F.R., Munksgaard, Copenhagen, p. 312

FORTRAN package to handle a direct access file

Riccardo Spagna
Istituto di Cristallografia - CNR, Sede di Monterotondo, Area della Ricerca Roma 1, Via Salaria Km.29, 00016 Monterotondo Stazione (Roma), Italy. E-mail: riccardo.spagna@ic.cnr.it

Introduction

This article is a description of a package in FORTRAN language, based on the original routines for the CRYSTALS program written by J.R. Carruthers (1977). By using this package, the programmer can handle a direct access disc file in which it is possible to go directly to a desired word, without considering the fixed length of the records. The technique uses a structured array in memory, so that the copies of records and some pointers are stored and managed by the routines.

Variables setting and opening the direct access file

First of all, we define some variables and the LINK array in COMMON blocks and then we have to open a direct access disc file. One important advantage of this sort of file is that it allows records to be read and written in any order. All records must have the same length and the record length has to be chosen when the file is created. We have the opportunity to choose any value, or find the recommended value by the computer manufacturer, or use the following FORTRAN 90 construct:

```
c
code for CheckRecordLength
      subroutine CheckRecordLength
! IOLENGTH= tells what record length is required for a given io-list.
      common/xdisc/iacc,ilss,iadd1,nfwd,nwd,nv,nwb,nwr,i,j,k,l,m,n,nnn
      dimension ivet(512)
      ivet(:) = 0
      inquire(IOLENGTH = nwr) ivet
      return
      end
```

The OPEN statement establishes a connection between a unit number and the disc file and it also establishes or verifies the properties of the file:

```
      open (unit= ncdfu, file= 'file_name', access = 'direct', recl= nwr)
```

Set up the user defined buffers

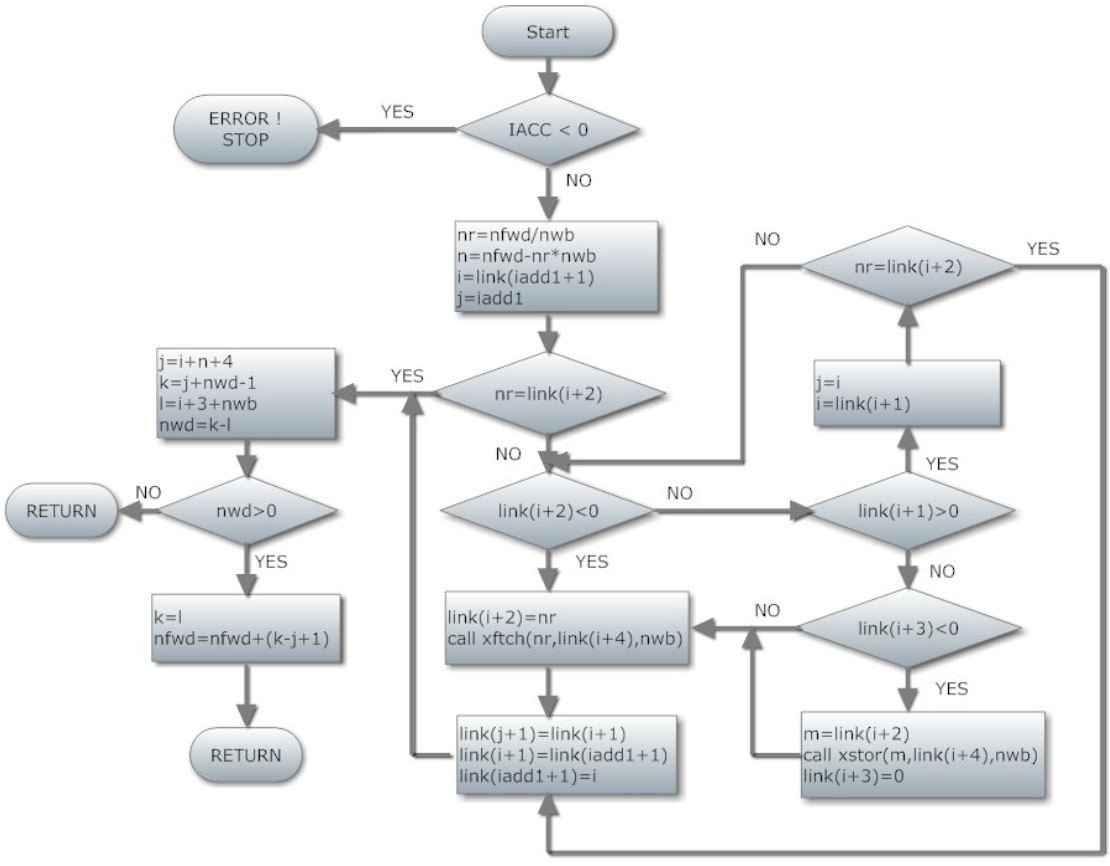
The LINK array has to be structured by requiring four pointers for each record's buffer. The routine STAR9 sets the start address and the last address to be used for the LINK array and calls the routine XGIV which sets up the buffers defined by the user (as shown in Figure 1). In the example, the number of words per buffer (nwb) is set to 512 and there are three buffers using a dimensioned array of 1600 words.

Figure 1) The coloured cells show the content of the LINK array as it results from XGIV routine.

	last rec used			previous address	next buffer or last	No.rec buffer	code	nwb words (record lenght)						
-10000	5			1	5+nwb+4	-10000	0							
1	2	3	4	5	6	7	8	9	...	...	...	...	...	4+nwb+4
				5	5+nwb+4+ nwb+4	-10000	0							
				5+nwb+4	6+nwb+4	7+nwb+4	8+nwb+4	9+nwb+4	...	...	...	...	...	4+nwb+4 +nwb+4
				5+nwb+4	-10000	-10000	0							
				5+nwb+4+ nwb+4	6+nwb+4+ nwb+4	7+nwb+4+ nwb+4	8+nwb+4+ nwb+4	9+nwb+4 +nwb+4	...	...	...	...	...	4+nwb+4 +nwb+4+ nwb+4

The I/O routines

Now, the programmer can use the routine XUP to write the data on the direct access file and XDWN to read the data when the start address of a data in the disc file is known. Both the subroutines call XFNDB which is the main routine to handle the buffers on memory and the records on disc file by calling the routines XFTCH and XSTOR. (See the flow chart below)



Imagine we wish to modify the file involving three records with record number NR1, NR2 and NR3. The updated content of the LINK array is shown in Figure 2) where the yellow cells are the starting codes, the pink cells are the control codes for each buffer and the blue cells contain the words of the records after the change made by the routine XUP.

For the j-th buffer, the address of the first control code is i=5+(j-1)*(nwb+4), then:

- LINK(i) contains the address of the next buffer,
- LINK(i+1) contains the address of the previous buffer, used to check the integrity of the chain of the I/O records,
- LINK(i+2) contains the record number of the buffer.
- the value -1 in LINK(i+3) advises that the record is modified (the value is set to zero when the buffer and the record are the same),

The value -10000 in LINK(i+1) means that this is the last buffer, and to satisfy the user request, the following operations have to be done:

1. the current buffer is written on disc,
2. LINK(i+3) is set to 0
3. the record NR4 is read from disc in the current buffer
4. the values of the address in LINK(i) and LINK(i+1) for all the buffer are changed to save the I/O order of the records.

LINK(2) contains the address of the last I/O record used.

Figure 2) The content of the LINK array after the I/O of three records..

	last rec used			previous address	next buffer or last	No.rec buffer	code	nwb words (record lenght)						
-10000	5+nwb+4 +nwb+4			1	-10000	nr1	-1							
1	2	3	4	5	6	7	8	9	...	...	...	...	...	4+nwb+4
				5	5	nr2	-1							
				5+nwb+4	6+nwb+4	7+nwb+4	8+nwb+4	9+nwb+4	...	...	...	...	...	4+nwb+4+ nwb+4
				5+nwb+4	5+nwb+4	nr3	-1							
				5+nwb+4+ nwb+4	6+nwb+4 +nwb+4	7+nwb+4 +nwb+4	8+nwb+4 +nwb+4	9+nwb+4 +nwb+4	...	...	...	...	...	4+nwb+4+ nwb+4+ nwb+4

Ancillary routine to write on disc

The programmer has to be aware that the modified buffers are in memory and they are written phisically on disc only if a new record has to be read and no free buffer is available. The routine XDUMP forces the buffers to be written to disc and the programmer calls it whenever he thinks it approriate.

Conclusions

The original package written by J.R. Carruthers (1977) and included in CRYSTALS program (Betteridge, P.W. *et al*., 2003) has worked very well for more than 35 years and the one I described here is part of CAOS (Camalli & Spagna, 1994), SIR (Burla, M.C. et al., 2005) and "IlMilione" (Burla, M.C. et al., 2007) programs. The routines are written in FORTRAN-77 and they should run on any computer with FORTRAN compiler.

List of the routines

```
c
code for bldsc
      block data bldsc
c--define block data for disc routines
c
      common/xbufd/link(1600)
      common/xdisc/iacc,ilss,iadd1,nfwd,nwdsk,nv,nwb,nwr,i,j,k,l,m,n,nnn
      data iacc/-1/,ilss/15/
      end
c
code for star9
      subroutine star9
c--initialize the disc routines
c
      li=1
      lf=1600
      call xgiv(li,lf,i)
      return
      end
c
code for xgiv
      subroutine xgiv(iadd0,nnl,llu)
c--set up the user defined buffers
c
c  iadd0  first free address in 'link' that can be used
c  nnl    length of user buffer area
c  llu    first word that may be used by the user (set on return)
c--
      common/xbufd/link(1600)
      common/xdisc/iacc,ilss,iadd1,nfwd,nwd,nv,nwb,nwr,i,j,k,l,m,n,nnn
c
c--set the number of words per block equal to the number per record
      nwb=nwr
c--calculate the number of buffers
      l=(nnl-4)/(nwb+4)
      iacc=-1
      llu=iadd0
      if(l) 2500,2500,2300
2300  continue
      l=min0(l,ilss)
      iadd1=iadd0
      link(iadd1)=-10000
      i=iadd1+4
      j=iadd1
      go to 2400
c--link the next buffer
2350  continue
      link(i)=j
      link(j+1)=i
      link(i+2)=-10000
      link(i+3)=0
      j=i
      i=i+nwb+4
      l=l-1
c--check if any more buffers can be link in
2400  continue
      if(l) 2450,2450,2350
c--set the last buffer flag
2450  continue
      link(j+1)=-10000
      iacc=1
      llu=i
2500  continue
```

```
      return
      end
c
code for xup
      subroutine xup(nfw,iadd,nw)
c--transfer from store to disc
c
c  nfw   address of first word to be used on disc
c  iadd  data array
c  nw    number of words to transfer
c--
      common/xbufd/link(1600)
      common/xdisc/iacc,ilss,iadd1,nfwd,nwd,nv,nwb,nwr,i,j,k,l,m,n,nnn
c--
      dimension iadd(nw)
c
c--transfer the arguments to the common block
      nfwd=nfw
      nwd=nw
      nv=1
c--check if there are any more words to transfer
1200  continue
      if(nwd)1250,1250,1300
1250  continue
      return
c--calculate the record number etc. for this transfer
1300  continue
      call  xfndb
c--write some numbers
      link(i+3)=-1
      do 2050 m=j,k
      link(m)=iadd(nv)
      nv=nv+1
2050  continue
      go to 1200
      end
c
code for xdwn
      subroutine xdwn(nfw,iadd,nw)
c--transfer from disc to store
c
c  nfw   address of first word to be used on the disc
c  iadd  data array
c  nw    number of words to transfer
c--
      common/xbufd/link(1600)
      common/xdisc/iacc,ilss,iadd1,nfwd,nwd,nv,nwb,nwr,i,j,k,l,m,n,nnn
c
      dimension iadd(nw)
c
c--transfer the arguments to the common block
      nfwd=nfw
      nwd=nw
      nv=1
c--check if there are any more words to transfer
1200  continue
      if(nwd) 1250,1250,1300
1250  continue
      return
c--calculate the record number etc. for this transfer
1300  continue
      call xfndb
c--read some numbers
      do 1950 m=j,k
      iadd(nv)=link(m)
      nv=nv+1
```

```
1950  continue
      go to 1200
      end
c
code for xfndb
      subroutine xfndb
c--find the required disc buffer
c
c--arguments are in the common block :
c
c  iadd1  first buffer word in 'link'
c  nfwd   current disc address
c  nwd    number of words left to transfer
c  nv     current position in the input or output array
c
c--the above variables are set on entry, and reset as necessary on exit.
c
c--the following variables are also set on exit :
c
c  i  address of the current buffer
c  j  address of the first word in the buffer for data
c  k  address of the last word in the buffer for data
c
c--the following variables must not changed :
c  l
c  m
c  n
c--
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
      common/xbufd/link(1600)
      common/xdisc/iacc,ilss,iadd1,nfwd,nwd,nv,nwb,nwr,i,j,k,l,m,n,nnn
c
c--check if the buffers have been allocated
      if(iacc)1050,1150,1150
c--attempt to read or write before declaring buffers
1050  continue
      write(ncwu,1100)
1100  format(42h1no buffers allocated to the disc routines)
      stop 46
c--calculate the record number etc. of transfers to be made
1150  continue
      nr =nfwd/ nwb
      n =nfwd-nr* nwb
c--search through the buffer list for free buffers and/or the buffer
c  containing this record
      i=link(iadd1+1)
      j=iadd1
c--check the first buffer to see if this is the correct one
      if(nr-link(i+2))1400,1750,1400
c--check if this is the correct buffer
1350  continue
      if(nr-link(i+2))1400,1700,1400
c--check for a free buffer wich also indicates the end of the stack
1400  continue
      if(link(i+2))1650,1450,1450
c--check if this is the last buffer
1450  continue
      if(link(i+1)) 1550,1550,1500
c--pick up the next buffer
1500  continue
      j=i
      i=link(i+1)
      go to 1350
c--rewrite the contents of the last buffer to the disc if nec.
1550  continue
      if(link(i+3)) 1600,1650,1650
```

```
1600  continue
      m=link(i+2)
      call xstor(m,link(i+4),nwb)
      link(i+3)=0
c--fetch the record of interest
1650  continue
      link(i+2)=nr
      call xftch(nr,link(i+4),nwb)
c--switch this buffer to the top of the buffer stack
c  all other buffers are forced one down the stack, so that the
c  least used one ends up at the end
1700  continue
      link(j+1)=link(i+1)
      link(i+1)=link(iadd1+1)
      link(iadd1+1)=i
1750  continue
c--calculate the number of words that can be transfered from this buffer
      j=i+n+4
      k=j+nwd-1
      l=i+3+nwb
      nwd=k-l
c--check if all the words are in this buffer
      if(nwd)1850,1850,1800
1800  continue
      k=l
      nfwd=nfwd+(k-j+1)
1850  continue
      return
      end
c
code for xdump
      subroutine xdump
c--rewrite all the buffers marked for writing to the disc
c
      common/xbufd/link(1600)
      common/xdisc/iacc,ilss,iadd1,nfwd,nwd,nv,nwb,nwr,i,j,k,l,m,n,nnn
c
c--check if any buffers are allocated
      if(iacc)2800,2550,2550
c--link into the first buffer
2550  continue
      i=link(iadd1+1)
      j=-1
c--check if the buffer required writing
2600  continue
      if(link(i+3))2650,2700,2700
c--write the buffer out
2650  continue
      j=link(i+2)
      call xstor(j,link(i+4),nwb)
      link(i+3)=0
c--check if this is the last buffer
2700  continue
      if(link(i+1)) 2800,2800,2750
2750  continue
      i=link(i+1)
      go to 2600
2800  continue
      return
      end
c
code for xftch
      subroutine xftch(i,j,k)
c--fetch data from the disc  -  this link controls the disc
c
c  i  disc address of the information in user records
```

```
c  j  array into which the data goes
c  k  number of words to read
c
c**this link is machine specific as it relates the
c  length of records seen by the user to the length
c  of hardware blocks on the disc
c
c--
      common/xdisu/ncdfu ,ndfle,irecx,ldaf,nbytes
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
      dimension j(512)
c
c**machine specific - relates the block length to the user record length
      n=i+1
      read(unit=ncdfu,rec=n,iostat=ier,err=1010)(j(m),m=1,k)
      return
1010  continue
      l=kwlnt(k)
      do 1015 m=1,l
      j(m)=0
1015  continue
      write(unit=ncdfu,rec=n,iostat=ier,err=1030)(j(m),m=1,l)
      return
1030  continue
      write(ncwu,1020) ier
1020  format(' xftch: ',i5,' error')
      stop 100
      end
c
code for xstor
      subroutine xstor(i,j,k)
c--store data on the disc - this link controls the disc
c
c  i  disc address of the information in user records
c  j  array containing the data
c  k  number of words to write
c
c**this link is machine specific as it relates the
c  length of records seen by the user to the length
c  of hardware blocks on the disc
c
c--the transfer is rounded up to fill one or more disc blocks
c
c--
      common/xdisu/ncdfu ,ndfle,irecx,ldaf,nbytes
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
      dimension j(512)
c
c**machine specific - relates block length to the user record length
      m=kwlnt(k)
      n=i+1
      write(unit=ncdfu,rec=n,iostat=ier,err=1010) (j(l),l=1,m)
      return
1010  continue
      write(ncwu,1020) ier
1020  format(' xstor: ',i5,' error')
      stop 200
      end
c
code for kwlnt
      function kwlnt(in)
c--assigns the number of words to complete a record
      common/xdisc/iacc,ilss,iadd1,nfwd,nwd,nv,nwb,nwr,i,j,k,l,m,n,nnn
      kwlnt=(in+nwb-1)/nwb*nwb
      return
      end
```

References

Betteridge, P. W., Carruthers, J. R., Cooper, R. I., Prout, K., Watkin, D. J. (2003). J. Appl. Cryst. 36, 1487.

Burla, M.C., Caliandro, R., Camalli, M., Carrozzini, B., Cascarano, G.L., De Caro, L., Giacovazzo, C.,Polidori, G., Spagna, R. (2005), *J. Appl. Cryst.* **38**, 381-388.

Burla, M.C., Caliandro, R., Camalli, M., Carrozzini, B., Cascarano, G.L., De Caro, L., Giacovazzo, C., Polidori, G., Siliqi, D., Spagna, R. (2007), *J. Appl. Cryst.* **40**, 609-613.

Camalli, M., Spagna, R. (1994), *J. Appl. Cryst*., **27**, 861-862.

Carruthers, J.R. (1977) IV ECM, Abstract Ob. 2

FORTRAN routines to create and use a Crystallographic Database

Riccardo Spagna
Istituto di Cristallografia - CNR, Sede di Monterotondo, Area della Ricerca Roma 1, Via Salaria Km.29, 00016 Monterotondo Stazione (Roma), Italy. E-mail: riccardo.spagna@ic.cnr.it

Introduction

J.S.Rollett (1970) described a way used in the *KDF9 Programs for crystal structure analysis* to store on disc the crystallographic information relating to a crystal structure analysis. He called a group of homogeneous crystallographic data as "LIST", for example cell parameters, symmetry, atomic parameters, reflections, and gave a NUMBER for each type of list. (See example below)

LIST TYPE	TITLE
1	Unit cell
2	Symmetry
3	Form Factors
4	Weighting Function
5	Atomic Parameters
6	Reflections
….	

Afterwards, J.R. Carruthers with Rollett and a wide group of programmers including myself wrote in FORTRAN the new program CRYSTALS for crystal structure analysis (Carruthers, 1977). This program improved the technique to save the lists on a direct access disc file by writing some routines to go directly to a desired word, without considering the fixed length of the records (Spagna, 2009a). A list index, or directory, was created to allow the program to fetch the crystallographic data in a very efficient way. This database was created for each new crystal structure analysis and subsequent runs of CRYSTALS could read and update the lists to perform crystallographic calculation.

On my return to Rome, I started to write the CAOS program (Cerrini & Spagna, 1977) by adapting the overall plan for the data lists of CRYSTALS to fit the HP21MX minicomputer with 32K, 16-bit word, of core memory (Spagna, 2009b). The routines to create and use the crystallographic database were a bit simplified by saving only the last version of each list and here I describe those used in a later version of CAOS program (Camalli & Spagna, 1994).

Variables setting and initialize the direct access file

First of all, we define some variables and the LINDEX array in a BLOCK DATA. The LINDEX array holds a copy in memory of the directory of the database.

Then the subroutine SETFI is called to initialize the disc file setting up the list directory. This is done in two steps. In the first one, the current values defined in the BLOCK DATA are saved in the preamble of NWBLO words. In the second step, the list directory is created by using the variables stored in the preamble. The subroutine which reads the directory will find the variables used to create the directory in the preamble, so it allows one to work with this database even if the next version of CAOS has different values defined in BLOCK DATA.

The scheme of the disc file is shown on Table 1), where four spaces are defined:

1) the EIA (Exchange Information Area), 1600 words, used in first version of CAOS and left unused in the later ones for compatibility;
2) the preamble block, 50 words, which holds the definition of some variables related to the database;

3) the directory of the database 240 words long (in this case), consisting in a table of six entries for 39 types of list plus six general values. The entries hold the information where the program can pick up the data related to each list;
4) the database of the crystallographic structure data grouped in lists, from location 1890 onwards (in this case).

Table 1). The direct access file.

Array in memory	Direct access file	Content
Exchange (1)	0	EIA
Exchange (2)	1	EIA
	...	...
	...	…
Exchange (1599)	1598	EIA
Exchange (1600)	1599	EIA
Lindex (1)	1600	Address of the first free position on the database. Initial value: 1600+240+50=1890
Lindex (2)	1601	Number of words of the table = 240
Lindex (3)	1602	Address of the table = 1600+50
Lindex (4)	1603	Number of words of the preamble =50
Lindex (5)	1604	Relative address of the structure title =11
Lindex (6)	1605	Number of words of each list entry = 6
Lindex (7)	1606	Not used = -10000
Lindex (8)	1607	Control code =10000
Lindex (9)	1608	Control code =10000
Lindex (10)	1609	Not used = -10000
Lindex (11)	1610	First word of the structure title (format (a4))
Lindex (12)	1611	….
……..		….
Lindex (30)	1629	Last word of the title
….		Not used = -10000.
…..		.....
Lindex (50)	1649	Not used = -10000
Lindex (1)	1650	Control code = 10000
Lindex (2)	1651	Maximum number of possible lists =240/6 – 1 = 39
Lindex (3)	1652	Start address of the table = 1600+50
Lindex (4)	1653	Last address of the table = 1600+50+240– 1
Lindex (5)	1654	Number of entry of each list = 6
Lindex (6)	1655	Table's length = 240

Lindex (7)	1656	1° entry of LIST 1
Lindex (8)	1657	2° entry of LIST 1
Lindex (9)	1658	3° entry of LIST 1
Lindex (10)	1659	4° entry of LIST 1
Lindex (11)	1660	5° entry of LIST 1
Lindex (12)	1661	6° entry of LIST 1
Lindex (13)	1662	1° entry of LIST 2
…	…	…
Lindex (235)	1884	1° entry of LIST 39
Lindex (236)	1885	2° entry of LIST 39
Lindex (237)	1886	3° entry of LIST 39
Lindex (238)	1887	4° entry of LIST 39
Lindex (239)	1888	5° entry of LIST 39
Lindex (240)	1889	6° entry of LIST 39
Store (nfl)	1890	First variable related to the first stored list
….	….	….
….	….	….

Entries for each N list (N is a number between 1 e 39) in the table having start address (i = N*6 +1) :
Lindex (i) list number = N
Lindex (i+1) list version number
Lindex (i+2) address on file of the first variable related to this list
Lindex (i+3) address on file of the last variable related to this list
Lindex (i+4) number of words of the preamble block of this list (or = 0)
Lindex (i+5) number of words of this list

Routines to read in memory the directory

The subroutine XLC is a generic routine to check if the LINDEX array was loaded in memory. If not, XCL calls SLDF to load the list directory in the LINDEX array, after verifying the integrity of the directory, by using the function JSLDF.

The two subroutines XNXTF and SNXTF read / write the address of the next free location in the database. This address is stored in the LFIRST position of the disc file.

Routines to read, write and print list entries

The subroutine SWLIN writes the entries for a list in the directory of the database by using the arguments associated with the corresponding actual arguments specified in the call:

ILN	List type
LSN	Serial number
LFW	Address of the first word of the list
LLW	Address of the last word of the list
LPB	Length of the preamble block for a block type list
LL	Length of the list

The list type must be a valid number (less or equal to the maximum possible value saved in LINDEX(2)) and the starting position of these entries in the LINDEX array is calculated as following:

I = ILN * LINDEX(5) + 1

where LINDEX(5) holds the number of entries for each list. LINDEX(3) is the starting address of the directory in the disc file and LINDEX(6) contains the length of the directory.

The subroutine SRLIN reads the entries for a list from the directory of the database and assigns these values to the argument specified in the call. The value LFW is set to negative if the list is not present on the database.

The routines SPRTF, XPRTH, XPRTL are used to print the content of the directory and KPRTH to print the entries for a list.

Routines to read and write a list in the database

When the programmer wants to save a list, he has to include some code, overall data and the crystallographic information related to that list in the large array STORE defined in the labeled COMMON/XDATA/ starting from the next free location of the array.

For example, the list of atomic parameters in this version of CAOS has the following format:

Relative Address in the array STORE	Description	Content
00	Security code	11111.0
01	List number	5.0
02	Serial number	
03		0
04 sys05	List code	-100.0
05	Increment for 1st overall parameter	27
06 md5o	Number of words for each overall parameter	1
07 n5o	Number of overall parameter	4
08 npre5	Length of list preamble	40
09 md5	Number of words for each atom	55
10 n5	Number of atoms	
11	Connectivity code	0/1
12	*not used*	
13	*not used*	
14 md5h	increment for the label of the atom	22
15 md5xyz	Increment for the positional parameter	6
16 Btot(1,1)	Orientation matrix	
17 Btot(1,2)		
18 Btot(1,3)		
19 Btot(2,1)		
20 Btot(2,2)		
21 Btot(2,3)		
22 Btot(3,1)		
23 Btot(3,2)		
24 Btot(3,3)		
25	Increment for the orientation	16

	matrix	
26 md5c	Number of words for each bond of the connectivity	3
27	1° overall parameter Scale	
28	2° overall parameter B(overall)	
29	3° overall parameter Extinction	
30	4° overall parameter Polarity	
31	σ(Scale)	
32	σ(B(overall))	
33	σ(Extinction)	
34	σ(Polarity)	
35	*not used*	
36	*not used*	
37	*not used*	
38	*not used*	
39	*not used*	
40 istore(l5)	Atomic species 1st atom (a4)	
41 istore(l5+ 1)	(species1) (a4)	
42 istore(l5+ 2)	(species2)	
43 istore(l5+ 3)	(species3)	
44 istore(l5+ 4)	(species4)	
45 store(l5+ 5)	Serial number	
46 store(l5+ 6)	X	
47 store(l5+ 7)	Y	
48 store(l5+ 8)	Z	
49 store(l5+ 9)	B[iso]	
50 store(l5+10)	b(1,1)	
51 store(l5+11)	b(1,2)	
52 store(l5+12)	b(1,3)	
53 store(l5+13)	b(2,2)	
54 store(l5+14)	b(2,3)	
55 store(l5+15)	b(3,3)	
56 store(l5+16)	Occ	
57 store(l5+17)	occ1 (for species1)	
58 store(l5+18)	occ2 (for species2)	
59 store(l5+19)	occ3 (for species3)	
60 store(l5+20)	occ4 (for species4)	
61 store(l5+21)	Code idev	0 / 1
62 istore(l5+22)	Name(1) (format a4) [Atomic label (4 words)]	
63 istore(l5+23)	Name(2)	
64 istore(l5+24)	Name(3)	
65 istore(l5+25)	Name(4)	
And so on…..	……	…….
95 istore(l5+md5)	Atomic species 2nd atom (a4)	
96 …..	…..	
97 …..		

nn-1=l5+n5*md5-1	last information for n5 atom	
nn = l5c istore(l5c)	number of bonds	
nn+1 istore(l5c+1)	atom1 (bond1)	Sequence number
nn+2 istore(l5c+2)	atom2 (bond1)	*idem*
nn+3 istore(l5c+3)	code (bond1)	0/1/-1
….	……	
….	……	
nn+istore(l5c)*md5c	last atom of last bond…..	

A call to the subroutine XXCLN sets the first 5 positions of the list and when the crystallographic information are included in the array STORE the routine XFRML writes the list on disc as following:

- call the routine SOWL which checks from the directory if the same list type is preset in the data-base;
- if yes, check if the new list is smaller than or equal the old one; in this case, the pointers of the new list are the same of the old one;
- if not, or the list is larger than the old one, the routine SOWL reads the next free location in the database using the routine XNXTF, sets this value as the starting address of the list, updates the next free location adding the length of the list and call the routine SNXTF which saves this value;
- increment the serial number of the list, write the list on disc by using the routine XUP;
- update the directory with the new entries for this list by using the routine SWLIN.

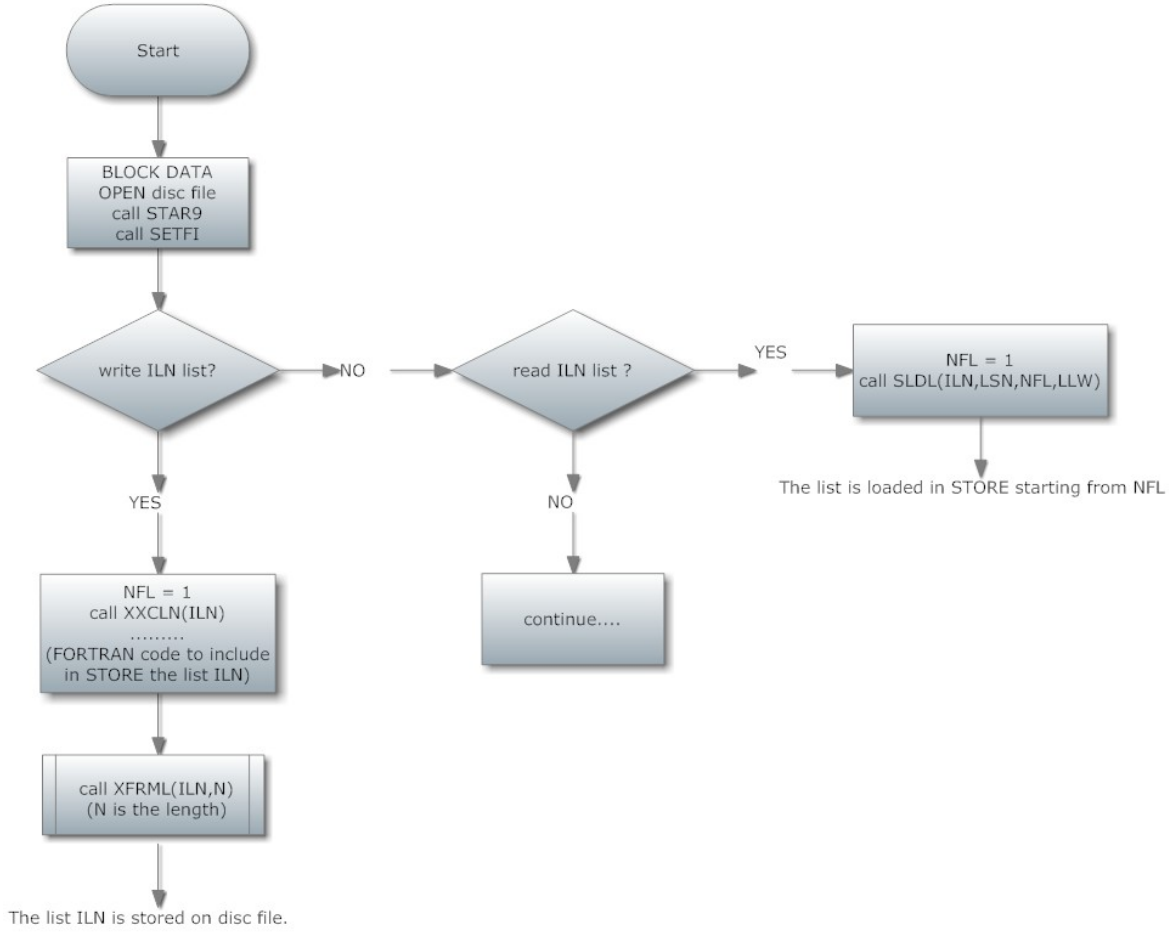
The routine SLDL loads the list from disc into the array STORE as following:

- call the routine SRLIN to read the entries of the list saved in the directory;
- obtain the starting address in STORE and call the routine SRDLS;
- this routine checks if the list exists in the database, if not, an error will displayed;
- if yes, by using the values saved in the entries, the list is read through the routine XDWN.

From the overall values of the list, the programmer defines a set of pointers in a labeled COMMON to access the data. In the example of list 5, the pointers in COMMON/LST05/ are the following:

l5	location of the first atom
m5	location of the current atom
md5	number of words per atom
n5	number of atoms
l5o	location of first overall parameter
m5o	location of the current overall parameter
md5o	number of words for each overall parameter
n5o	number of overall parameters
l5d	starting address of list 5
n5d	length of list 5
l5c	location of the first bond
md5c	number of words for connectivity
l5ori	location of first value of orientation matrix (9 parameters)
md5h	increment for label
md5xyz	increment for coordinates

A simple flow chart and source code of a program which writes/reads a list is shown below




```
c
code for main
      program main
c--program to test the routines
c
      common/xdisu/ncdfu ,ndfle,irecx,ldaf,nbytes
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
      common/ylimi/nfl,lfl,nl,nflmx,lflmx,nlmx
      common/xdisc/iacc,ilss,iadd1,nfwd,nwd,nv,nwb,nwr,i,j,k,l,m,n,nnn
c
      common/xdata/ store(800000)
      dimension istore(800000)
      equivalence (store,istore)
c
      character*60 ititle
      common/compd/ititle
c
      character*40 FileName
      character*10 alfa
      data alfa/'abcdefghil'/
c
      ncru=5
      ncwu=6
      ncdfu=21
      FileName='FileTest.bin'
      ititle='Default title'
      call CheckRecordLength
c--open the direct access disc file
      open (unit= ncdfu, file = FileName, access = "direct", recl= nwr)
      call star9
      call setfi
      call xlc
      nfl=1
```

```
c--form a list 1
      iln=1
      call xxcln(iln)
      init=nfl+5
      do i=1,10
         istore(init)=i
         init=init+1
      enddo
      call xfrml(iln,15)
c--form a list 2
      iln=2
      call xxcln(iln)
      init=nfl+5
      read(alfa,'(10a1)') (istore(l),l=init,init+9)
      call xfrml(iln,15)
c--print the file index
      call sprtf
c--read list 1 and 2
      nfl=1
      istore(1:100)=0
      iln=1
      call sldl(iln,lsn,lfw,llw)
      write(ncwu,1000)(store(l),l=lfw,lfw+4),(istore(l),l=lfw+5,llw)
1000  format(1x,5f10.0,/,1x,20i5)
      iln=2
      call sldl(iln,lsn,lfw,llw)
      write(ncwu,1010)(store(l),l=lfw,lfw+4),(istore(l),l=lfw+5,llw)
1010  format(1x,5f10.0,/,1x,20a1)
      stop
      end
```

List of the routines

```
c
code for blolist
      block data blolist
c--define block data for list routines
c
      common/xfile/ifile,ilist,iend,infle,nwfle,icfle,lid(4),ifine
      common/xfili/lfirst,nwblo,lindex(240)
c
      data nwblo/50/,lfirst/1600/
      data infle/240/,nwfle/6/,icfle/-1/
      data ifile/10000/,ilist/11000/,iend/11111/,ifine/-11111/
      end
c
code for setfi
      subroutine setfi
c--initialize the direct access file
c
      character*60 ititle
      common/compd/ititle
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
      common/xfile/ifile,ilist,iend,infle,nwfle,icfle,lid(4),ifine
      common/xfili/lfirst,nwblo,lindex(240)
c
c--find the initial constants
      do 990 i=1,nwblo
      lindex(i)=-10000
990   continue
```

```
      i=infle
      j=nwblo
      k=j+i
      lindex(1)=lfirst+k
      lindex(2)=i
      lindex(3)=lfirst+j
      lindex(4)=j
      m=11
      lindex(5)=m
      lindex(6)=nwfle
      lindex(8)=ifile
c--next words is to indicate the last version of caos
      lindex(9)=ifile
      read(ititle,1000)(lindex(l),l=m,m+14)
1000  format(15a4)
      call xup (lfirst,lindex,j)
      m=infle/nwfle-1
      lindex(1)=ifile
      lindex(2)=m
      lindex(3)=lfirst+j
      lindex(4)=lfirst+k-1
      lindex(5)=nwfle
      lindex(6)=i
      l=nwfle+1
      do 1010 i=1,m
      do 1010 k=1,nwfle
      lindex(l)=0.0
      l=l+1
1010  continue
      i=lindex(6)
      call xup (lindex(3),lindex,i)
      call xdump
      call sldf
      return
      end
c
code for xlc
      subroutine xlc
c--initial subroutine to load all the file index information
c
      common/xfile/ifile,ilist,iend,infle,nwfle,icfle,lid(4),ifine
c
      if(icfle)1000,1010,1010
1000  continue
      call sldf
      icfle=1
1010  continue
      return
      end

c
code for sldf
      subroutine sldf
c--load the  list directory
c
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
c
      if(jsldf(in)) 1010,1005,1005
1005  continue
      return
1010  continue
c--error because the file index has been corrupted
      write(ncwu,1020)
1020  format(34h the file index has been corrupted)
      stop 1111
      end
```

```
c
code for jsldf
      function jsldf(in)
c--load the details of the file index when beginning to use a file
c
      character*60 ititle
      common/compd/ititle
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
      common/xfile/ifile,ilist,iend,infle,nwfle,icfle,lid(4),ifine
      common/xfili/lfirst,nwblo,lindex(240)
c--
      jsldf=0
      call xdwn(lfirst,lindex,10)
      i=lindex(4)
      call xdwn (lfirst,lindex,i)
      j=lindex(5)
      write(ititle,1000)(lindex(k),k=j,j+14)
1000  format(15a4)
      i=lindex(2)
      call xdwn (lindex(3),lindex,i)
c--check that the file index is correctly set up or  has been
c  corrupted
      if(lindex(1)-ifile) 1010,1005,1010
1005  continue
      return
1010  continue
      jsldf=-1
      go to 1005
      end
c
code for xnxtf
      subroutine xnxtf(ll)
c—read the next free position
c
      common/xfili/lfirst,nwblo,lindex(240)
      dimension  locc(1)
      call xdwn(lfirst, locc,1)
      ll= locc(1)
      return
      end
c
code for snxtf
      subroutine snxtf(ll)
c—write the value of “ll”  as the new next free position
c
      common/xfili/lfirst,nwblo,lindex(240)
      dimension  locc(1)
      locc(1)=ll
      call xup(lfirst,locc,1)
      call xdump
      return
      end
c
code for swlin
      subroutine swlin (iln,lsn,lfw,llw,lpb,ll)
c—write the entries for a newly created list into the file index
c
c  iln  the list type number
c  lsn  the list serial number
c  lfw  first word accupied by the list
c  llw  last word occupied by the list
c  lpb  the I of the preamble block for a block type list
c  ll   list I
c
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
      common/xfili/lfirst,nwblo,lindex(240)
```

```
c
      call xlc
      if (iln) 1000,1000,1010
1000  continue
      write(ncwu,1001) iln
1001  format(///,i7,25h is not a valid list type)
      stop  1113
1010  continue
      mln=lindex(2)
      il =lindex(5)
      if(iln-mln) 1030,1030,1000
1030  continue
      i=iln*il+1
      lindex(i  )=iln
      lindex(i+1)=lsn
      lindex(i+2)=lfw
      lindex(i+3)=llw
      lindex(i+4)=lpb
      lindex(i+5)=ll
      j=lindex(6)
      call xup (lindex(3),lindex,j)
      if(istat2.le.2) return
      call xprth
      call xprtl(lindex(i),il)
      return
      end
c
code for srlin
      subroutine srlin (iln,lsn,lfw,llw,lpb,ll)
c—read the entries  for a list in the current index table
c
c  iln  the list type number
c  lsn  the list serial number
c  lfw  first word occupied by the list
c       'lfw' is set negative if no such list exists
c  llw  last word occupied by the list
c  lpb  the I of the preamble block for a block type list
c  ll   list I
c
      common/xfili/lfirst,nwblo,lindex(240)
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
c
      if(iln)1020,1020,1000
1000  continue
      call xlc
      mln=lindex(2)
      il =lindex(5)
      if(iln-mln) 1010,1010,1020
1010  continue
      i=iln*il+1
      if(lindex(i)) 1020,1020,1040
1020  continue
      lfw=-1
1030  continue
      if(istat2.le.2) return
      call xprth
      call xprtl(lindex(i),il)
      return
1040  continue
      lsn=lindex(i+1)
      lfw=lindex(i+2)
      llw=lindex(i+3)
      lpb=lindex(i+4)
      ll=lindex(i+5)
      go to 1030
      end
```

```
c
code for xxcln
      subroutine xxcln(in)
c—set on the store the first values of the list “in”
c
      common/ylimi/nfl,lfl,nl,nflmx,lflmx,nlmx
      common/xfile/ifile,ilist,iend,infle,nwfle,icfle,lid(4),ifine
      common/xdata/ store(800000)
c
      store(nfl  )=float(iend)
      store(nfl+1)=float(in)
      store(nfl+2)=1.0
      store(nfl+3)=0.0
      store(nfl+4)=-100.0
      return
      end
c
code for xfrml
      subroutine xfrml (iln, n)
c—form a list
c
c  iln list number
c  n   the length of the list
c
c  the list is assumed to begin at “nfl”
c
      common/ylimi/nfl,lfl,nl,nflmx,lflmx,nlmx
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
      common /xdata/ store(800000)
      dimension istore(800000)
      equivalence (store,istore)
c
      ln=n
      call sowl(ln,iln,lsn,lfw,llw,lpb,ll)
      if (iln) 1000,1000,1030
1000  write(ncwu,1010) iln
1010  format(/,i7,’ is not a valid list type’)
      return
1030  continue
      store(nfl+2)=lsn
      call xup(lfw,istore(nfl),n )
      call swlin(iln,lsn,lfw,llw,lpb,ln)
      call xdump
      return
      end
c
code for sowl
      subroutine sowl  (lcl,iln,lsn,lfw,llw,lpb,ll)
c—write or overwrite a list on the disc
c
c  lcl  length that has to be written
c  iln  list number
c  lsn  list serial number
c  lfw  location of first word on the disc
c  llw  location of the last word on the disc
c  lpb  length of the preamble block of the list to be overwritten
c  ll   list length
c
c—find the list of this type
      lsn=0
      call srlin (iln,lsn,lfw,llw,lpb,ll)
      if(lfw) 1020,1020,1010
c—check if the current list is of the right length
1010  continue
      if(ll-lcl) 1020,1060,1060
c—list can not be overwritten
```

```
c—find the next free location
1020  continue
      call xnxtf(ll)
      lfw=ll
      llw=lfw+lcl
      call snxtf(llw)
      llw=llw-1
      lpb=0
1060  continue
      ll=lcl
      lsn=lsn+1
      return
      end
c
code for sldl
      subroutine sldl (iln,lsn,lfw,llw)
c—load list into store
c  iln  list type number
c  lsn  list serial number
c  lfw  location of the first word of the list in store
c  llw  location of the last word of the list in store
c
c—the  list is brought up from the bottom of the store,
c  “lfw” and “llw” are set on return
c
      common/ylimi/nfl,lfl,nl,nflmx,lflmx,nlmx
c
      call srlin (iln,lsn,li,lj,lk,ll)
      lfw=nfl
      llw=lfw+ll-1
      nfl=llw+1
      call srdls(iln,lsn,lfw,l)
      return
      end
c
code for srdls
      subroutine srdls(iln,lsn,lfw,lll)
c—read a list from the disc
c
c  iln  list type number
c  lsn  list serial number — set on return
c  doc  array to hold the list
c  lll  number of words to bring down — if zero the whole list
c       is brought down,and@ll@ is set to the number of words read
c
c—for lists with a  reamble block,only the preamble block is read,
c  and “lll” is set negative on return
c
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
      common /xdata/ store(800000)
      dimension istore(800000)
      equivalence (store,istore)
c
      call srlin (iln,lsn,li,lj,lk,ll)
      if(li) 1000,1000,1020
1000  continue
      write(ncwu,1010) iln
1010  format(///,16h no list of type,i5,8h  stored,/)
1015  continue
      stop 1114
1020  continue
      if(lk) 1040,1040,1030
1030  continue
      ll=lk
1040  continue
      lll=ll
```

```
      nll=lll
      call xdwn(li,istore(lfw),nll)
      il=store(lfw+1)
      if(il-iln)1061,1063,1061
1061  continue
      write(ncwu,1062)iln
1062  format(//,' requested list',i4,' has been corrupted')
      go to 1015
1063  continue
      if(lk) 1080,1080,1070
1070  continue
      lll=-lll
1080  continue
      return
      end
c
code for sprtf
      subroutine sprtf
c--print the complete file index
c
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
      common/xfili/lfirst,nwblo,lindex(240)
c
      dimension iq(10)
      data iq(1)/2h i/,iq(2)/2hnd/,iq(3)/2hex/,iq(4)/2h o/
      data iq(5)/2hf /,iq(6)/2hfi/,iq(7)/2hle/,iq(8)/2h l/
      data iq(9)/2his/,iq(10)/2hts/
c--
      call xlc
1000  continue
      mln=lindex(2)
      il =lindex(5)
      write(ncwu,1010)(iq(i),i=1,10),lindex(3),lindex(6)
1010  format(/,6h print,10a2,' ( ',7haddress,i8 ,' , ',
     2 6hlength,i8,' )')
      call xprth
      i=il+1
      do 1030 j=1,mln
      if (lindex(i)) 1025,1025,1020
1020  continue
      call xprtl(lindex(i),il)
1025  continue
      i=i+il
1030  continue
      call xnxtf(ll)
      write(ncwu,1035) ll
1035  format(/,' next free location :',i8)
      return
      end
c
code for xprth
      subroutine xprth
c--print the heading for an index entry
c
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
c--
      write(ncwu,1000)
1000  format(/,10h list type, 4x,
     2 10hserial no.,7x,
     3 7haddress,7x,
     4 15hpreamble length,4x,
     5 12htotal length,/)
      return
      end
c
code for xprtl
```

```
      subroutine xprtl (lid,il)
c--print an index entry which is given 'id'
c
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
c--
      dimension lid(*)
c--
      write(ncwu,1000)(lid(i),i=1,il)
1000  format(i7  ,i13  ,i11  ,2x,2hto,i8  ,i11  ,i18  )
      return
      end
c
code for kprth
      function kprth(iln)
c--print the list heading for current list 'ln'
c
c--return values of the function are as follows:
c  -1 no such list stored
c   0 non-blocked list stored
c   1 blocked list of this type stored
c--
      common/xunit/ncru,ncwu,ncpu,ncpt,itty,istat2,iabort,intty
c--
      kprth=0
      call srlin(iln,lj,lk,ll,lm,ln)
      if(lk)1010,1010,1040
1010  continue
      write (ncwu,1020) iln
1020  format(' no list',i4,' stored',/)
      kprth=-1
1030  continue
      return
1040  continue
      if(istat2.ne.0) write(ncwu,1050)iln,lj,lk,ln
1050  format(/,' print list',i4,
     1 '( serial no.',i4,', address',i8,' , length',i8,' )')
      if(lm) 1030,1030,1060
1060  continue
      kprth=1
      go to 1030
      end
```

References

Camalli, M., Spagna, R. (1994), *J. Appl. Cryst.*, **27**, 861-862.

Carruthers, J.R. (1977) IV ECM, Abstract Ob. 2

Cerrini, S., Spagna, R. (1977) IV ECM, Abstract A212

Rollett, J.S. (1970) *Crystallographic Computing*, Edited by Ahmed F.R., Munksgaard, Copenhagen, p. 302

Spagna, R. (2009a) *Commission on Cryst. Comp., IUCr Newsletter*, No. 10, 97

Spagna, R. (2009b) *Commission on Cryst. Comp., IUCr Newsletter*, No. 10, 92

PLATON: Past, Present and Future

Ton Spek
National Single Crystal Service Facility, Utrecht University, H.R. Kruytgebouw, N-801, Padualaan 8, 3584 CH Utrecht, the Netherlands. E-mail: a.l.spek@uu.nl ; http://www.platonsoft.nl/

PLATON is a multipurpose single crystal structure analysis tool, written in the FORTRAN language, with a development history of 30 years. It collects the experience and know-how that was gathered doing structure determinations for over more than 40 years. PLATON is likely most used outside Utrecht as part of the IUCr CheckCIF facility but also for its unique SQUEEZE, ADDSYM and TwinRotMat tools. This contribution is meant to offer some insight in the evolutionary process of its creation.

How it all started

The program suite that is currently designated as PLATON started its life around 1980 as a program named PLATO. The terminal N was added at a later stage in order to differentiate a significantly New version of the program from an earlier version. Some see my first name 'Ton' in it but that was unintentional. Most of my programs at that time were given Greek names such as HELENA for our data reduction program of Enraf-Nonius CAD4 diffraction data. The name PLATO was chosen as a companion to the related molecular graphics program PLUTO. PLATO was modeled on the geometry program GEOM that came with the Cambridge Crystallographic Database distribution. PLATO was meant to replace an earlier suite of programs that was in use in Utrecht at that time for molecular geometry calculations. That earlier suite was based on software written in the ALGOL language, the language of the Electrologica X8 university single user mainframe, and not very suitable for further development on the new multi-user Control Data/FORTRAN university platform.

PLATO(N) is developed in the context of our national single crystal service facility that started around 1971. There was a need, in view of the growing number of studies, for the automatic generation of tables with bond distances, bond angles and torsion angles, all with associated standard uncertainties. Subsequently, this geometry set was extended with automatic ring search and puckering analysis, least squares planes, hydrogen bonds etc. In this way, clients could be supplied with an extensive report with hopefully everything they might want to know about their structure.

PLATON was originally designed to work in conjunction with a modified version of SHELX76 through RES files. With the introduction of SHELX-93 onwards, CIF was the main interface for data exchange between both programs.

Hardware platforms came and went. The first one was the Control Data CDC6400 Fortran platform of Utrecht University for which it was essential that the executable was not larger than 65000 words of 60 bits. This called for very 'word-efficient' programming. Traces of that requirement can still be found in the current source. Heaven came around 1985 with the introduction of Digital Equipment microVAX's in our research group with a capacity of about 1/8th each of the central university mainframe. A next generation in the early 1990's formed DEC-UNIX workstations that were already a factor of 20 faster. Eventually development continued on the INTEL/Linux platform.

Molecular Graphics

Parallel with the development of PLATO(N) there was a similar development of a graphics program (PLUTON) that was inspired by a program named PLUTO (Bill Clegg version). PLUTON was later on made part of PLATON and so was some code from the well known program ORTEP for the display of displacement ellipsoids. Drivers were written for Tektronix and HP graphics terminals and Calcomp and HP pen plotters. Graphics standards changed over time. The current graphics is based on X11-Windows for display and PostScript for hardcopy output. HPGL is available as an alternative format. All graphics

calls go through one subroutine that contains the hardware dependent graphics code. A change of graphics platform to e.g. MS-Windows will call only for an adaptation of that part of the code.

Tools

Over time, numerous tools were added. Some were adapted from existing software such as the code for TLS analysis by Y. Shmueli or the MISSYM algorithm for missed symmetry detection by Y. LePage. Other tools were developed from scratch in order to solve our structure analysis problems. Examples are the SQUEEZE tool to handle the contribution of disordered solvent in the structure refinement and TwinRotMat for the automatic detection of missed twinning. One of the most recent additions is an implementation of the Charge Flipping algorithm (Oszlanyi & Suto) for structure solution. Also a new tool, the Hooft parameter, was introduced as an alternative for absolute structure determination based on the refinement of the Flack parameter.

Around 1990, work was started to automate routine structure determinations. That tool, SYSTEM-S, included space group determination, structure determination and refinement including the introduction of hydrogen atoms. This tool can be run in either fully automatic mode or in a guided mode in which next-step suggestions could be overruled with alternative instructions. Examples are a different choice of space group and different structure solution method. The program handles the otherwise tedious I/O for the various programs that can be called on demand such as SHELXS, DIRDIF or SIR for structure determination. Also PLATON is called to carry out various functions. The SYSTEM-S tool is currently part of the main PLATON code.

Structure Validation

The introduction of the CIF standard for data exchange and archival opened the possibility to check the data for completeness and consistency. The obvious next step was a request by Syd Hall, co-author of the CIF standard, who was at that time section editor of Acta Cryst. Section C, to investigate the possibility to automatically check CIF's for issues such as missed voids in a structure and missed higher symmetry. This effort eventually resulted in the currently implemented IUCr CheckCIF facility for structure validation that is now used for virtually every structure that is part of a publication. For that reason, PLATON is likely the next frequently used program after SHELXL.

A recent addition to the CheckCIF facility is the validation of the reflection data on which the structure analysis is based. This includes a report on completeness of the data set, resolution, missed reflections and consistency with the derived data in the CIF.

Detailed analysis of difference density maps for spurious peaks is possible with contoured Fourier maps.

Implementation

Software development is currently done on the Linux platform. The Linux version of PLATON consists of a single Fortran source with currently 136000 lines of code and comment and a small C program that serves as an interface to the X-Windows graphics system. The program source compiles both on the INTEL/LINUX and INTEL/MAC-OSX platform. The only external dependency is on the X11 graphics library. The SYSTEM-S tool requires that at least SHELXL is present in the environment. The MS-Windows version (excluding the SYSTEM-S tool, that is not included on that platform) is maintained by Louis Farrugia from Glasgow University.

Input files are generally the .RES or CIF files from SHELXL for the derived parameters and HKL or FCF for the reflection data. Output can be RES, CIF, PDF, HKL or FCF style. Printable output is in LIS or PostScript format.

Instructions can be given in i) the operating system command line (see Appendix 1), ii) the PLATON command window, or iii) by left clicking in various menu's. HELP is available by right clicking on menu items when the HTML help tree is either available locally or over the internet. Various program functions (such as structure validation) are also available through switches on the command line and can thus be called easily from other programs.

Fig. 1 shows the PLATON opening menu with all clickable tools on the main menu and various options on side menu's.

Fig.1: *Opening Window of* ***PLATON*** *with the various clickable tools.*

Future development

The development of PLATON is largely event driven, either on the basis of our own needs and progressing knowledge or on the basis of the highly valued suggestions by outside users. Already available tools often suggest extensions. New applications and insights often call for the addition of more code. Obsolete code, such as drivers for graphics media that are no longer used, is removed and new code added to handle the new applications. Original Fortran66 specific code has been replaced in the past by Fortran77 code and will soon be upgraded to the Fortran95 standard.

The button 'STRUCTURE?' on the menu in Fig. 1 represents a very early version of a new automatic structure determination tool based on Charge Flipping that might supersede eventually the SYSTEM-S tool that is UNIX platform specific.

One of the design features of PLATON is to be as independent as possible from particular hardware and libraries. This makes the migration to new platforms relatively simple. The price is that PLATON lacks

menu's based on popular Widget-sets. However, the design of PLATON makes it possible to create an external Widget-set based tool that draws on functionality available in PLATON.

The Age-concern issue

Programs such as SHELX, ORTEP and PLATON are single author efforts. They can survive technically as long as there is a FORTRAN compiler available for the target platform. Changes in the environment (e.g. Input/Output) are usually easily implemented. However, science might move on. The implementation of new ideas might be nearly impossible when they require the change of basic design features. An example is the single number symmetry code (ORTEP inspired) associated with an atom. No cell translations can be represented with it outside the range -5 to +4.

Eventually, a complete rewrite might be needed on the basis of the wealth of experience coded in the current software. Efforts are underway to document the various algorithms that are implemented.

More Information on PLATON

Validation: A.L.Spek (2009). Acta Cryst. D65, 148-155.

Information on PLATON can be found at http://www.platonsoft.nl/

Editorial addition - Appendix 1: “How-to-run Platon from the command line” on following page

Appendix 1: How-to-run Platon from the command line

via X-ray Diffraction Laboratory, Department of Chemistry, Texas A & M University, March 31, 2009
http://www.chem.tamu.edu/xray/pdf/notes/how_to_run_platon_cl.pdf

Platon.exe should be placed in the PATH (Windows and Unix).

At the command line type platon -switch project_name.(ins, .fcf,.spf,.cif)

For example to run an interaction twinrotmat in platon from the command line employing the *.fcf file type

platon -L project_name.fcf

See below for more switch options

'-' - No data from file Read (Switch to I/O window)
'-a' - ORTEP/ADP [PLOT ADP]
'-b' - CSD-Search [CALC GEOM CSD]
'-c' - Calc Mode [CALC]
'-d' - DELABS [CALC DELABS]
'-e' - MULABS
'-f' - HFIX
'-g' - GenRes-filter [CALC GEOM SHELX]
'-h' - HKL-CALC [ASYM GENERATE]
'-i' - Patterson PLOT
'-j' - GenSPF-filter [CALC GEOM EUCLID]
'-k' - HELENA
'-l' - ASYM VIEW
'-m' - ADDSYM (MISSYM) [CALC ADDSYM]
'-n' - ADDSYM SHELX
'-o' - Menu Off
'-p' - PLUTON Mode
'-q' - SQUEEZE [CALC SQUEEZE]
'-r' - RENAME (RES)
'-s' - SYSTEM-S
'-t' - TABLE Mode [TABLE]
'-u' - Validation Mode [VALIDATION]
'-v' - SOLV Mode [CALC SOLV]
'-w' - Difference Map Plot
'-x' - Fo-Map PLOT
'-y' - SQUEEZE-Map PLOT
'-z' - WRITE IDENT
'-A' - PLATON/ANIS
'-C' - GENERATE CIF for current data set (e.g. .spf or .res)
'-F' - SILENT NQA SYSTEM-S PATH (FILTER)
'-I' - AUTOFIT 2 MOLECULES
'-K' - CALC KPI
'-L' - TWINROTMAT (INTERACTIVE)
'-M' - TWINROTMAT (FILTER MODE)
'-N' - 'ADDSYM EQUAL SHELX' MODE
'-O' - PLOT ADP (PostScript)
'-P' - Powder Pattern from Iobs
'-Q' - Powder Pattern from Icalc
'-R' - Auto Renumber and Write SHELX.res
'-S' - CIF2RES + FCF2HKL filter
'-T' - TwinRotMat
'-U' - CIF-VALIDATION (without VALIDATION DOC)
'-V' - FCF-VALIDATION (LAUE)
'-W' - FCF-VALIDATION (BIJVOET)
'-X' - Stripped SHELXS86 (Direct Methods Only) Mode
'-Y' - Native Structure Tidy (Parthe & Gelato) Mode
'-' - No Input File Assumed

List of programs archived at Armel Le Bail's Crystallography Source Code Museum

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

(# : mixed languag Fortran + C)

FORTRAN

1960s	1970s	1980s	1990s	2000s
tracer-65	fordap-70	exfft-80	treor-90	espoir-00
patmat-67	reduce-70	multan-80	dicvol-91	ortep-00
cryls-68	ortep-71	normal-80	powder-91	xtal-00
datap-68	sadiana-71	search-80	rmca-92	diffax-01
lsqpl-68	wilson-72	dicvol-82	pluto-92	
orfls-69	xanadu-73	xrs-82	sir-92#	
weight-69	agnost-74	mprof-83	sirpow-92#	
	icon-74	ito-84	stidy-92	
	linex-74	orffe-84	tmacle-92	
	camel-75	struplo-84	xyz-92	
	latcon-75	treor-84	absorb-93	
	dls-76	block-85	parst-93	
	shelx-76	exfft-85	sbgbbg-93	
	camel-77	getspec-85	strumo-93	
	lazy-77	geom-85	aritve-94	
	louv-77	lsq-85	dbws-94	
	xfls-77	ortep-85	cascade-95	
	celref-78	rbls-85	difabs-95	
	fordap-79	search-85	gtsym-95	
	shelx-79	sort-85	ccsl-96	
	volcal-79	wtanal-85	lhpm-96	
		hole-86	mprof-96	
		powd12-86	ortep-96	
		shadow-86	struvir-96	
		shelxs-86	caos-97	
		visser-86	cif2sx-97	
		appleman-	difrac-97	
		ito-87	dirdif-97	
		patsee-87	fullprof-97	
		pawley-87	glassvir-97	
		wppf-87	laue-97	
		dragon-88	tessel-97	
		mcmag-88	xhydex-97	
		pro-fit-88	iucrval-98	
		xlat-88	lapod-98	
		dbw-89	thma-98	
		fullprof-89	espoir-99	
		rietan-89	hydrogen-	
		pluto-??	promet-99	

OTHER (C, C++...)

1980s	1990s
	babel-94
	raswin-94
	xnd-95
	atominfo-96
	crystal-96
	drawxtl-96
	sginfo-96
	zefsall-99

Some Photographs and Reminiscences from the Computing Schools of the 1970's

Ton Spek
Director of National Single Crystal Service Facility, Utrecht University, H.R. Kruytgebouw, N-801, Padualaan 8, 3584 CH Utrecht, the Netherlands. E-mail: a.l.spek@uu.nl

NATO Crystallography School University of York, UK 1971

York was in 1971 one of the most active centres of Direct Methods development, The program MULTAN, that remained the standard for structure determination by Direct Methods for 15 years, was developed mainly there by Michael Woolfson and Peter Main. The school, sponsored by NATO, gave a broad overview of the theory and methods of structure determination. The medieval style of the conference dinner (eating with your hands) will be in the memory of all who attended.

Fig. 1: *Participants at the 1971 NATO Crystallography School. Some of the participants are: Front left-right: Woolfson, Ewald, Rogers. Front right to left next to the two ladies: Paul Beurskens and Davide Viterbo. I am two rows after the lady in the front/middle. Behind Rogers is Bill Duax.*

Extra identities via David Watkin: Front row right-2 Jorun Sletten (Norway). 3rd row - extreme right (David Watkin), right-2 Chui (no one ever knew his other name). Back row, just left of central white pilar, John Rollett

Extra identities via the Editor: between front left 4 and 5, Kirsten Peterson; above left of Kirsten Peterson is Peter Main; above left of Peter Main is Bob Gould.

Direct Methods in X-ray Crystallography 1973, Parma, Italy: Lectures by H. Hauptman

This meeting was organized by Prof. Nardelli in Parma in the beautiful spring of 1973. Dr Hauptman, (then on a three months scientific leave c/o the Istituto di Mineralogia at the Università di Bologna), gave in his lectures a complete overview of the theories that he developed for structure determination by Direct Methods and for which he later received the Nobel Prize together with Jerome Karle. Lectures were in the morning and late afternoon, separated by the traditional long siesta.

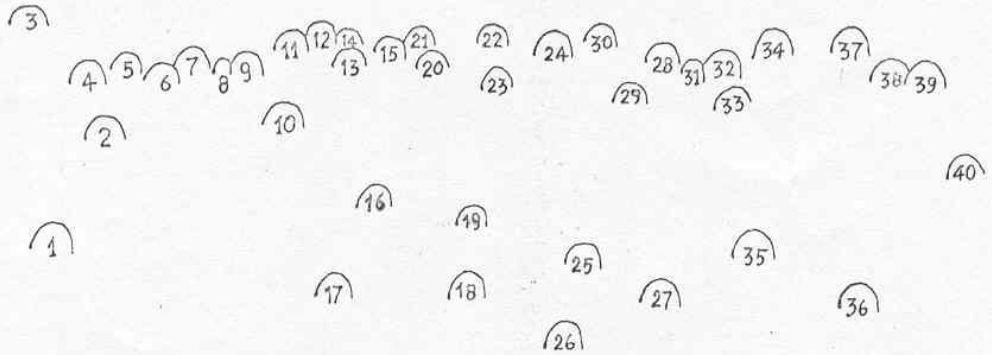
Fig. 2a: *Participants at the Direct Methods in X-ray Crystallography 1973, Lectures by H. Hauptman. Legend listing participants in Fig 2b overleaf.*

DIRECT METHODS IN X-RAY CRYSTALLOGRAPHY

by

Herbert HAUPTMAN

Parma 2-14 April 1973



1 P.T. Beurskens	15 H. Boller	28 G.D. Andreetti
2 L. Cavalca	16 A. Gaetani	29 B. Kojic-Prodic
3 M. Nardelli	17 G. Chiari	30 H.V. Gabrielsen
4 A. Corradi	18 H. Schenk	31 P. Domiano
5 A. Mangia	19 A. Chiesi	32 R.O. Gould
6 C. Palmieri	20 C.J.E. Kempster	33 E. Bang
7 G. Bocelli	21 S. French	34 D. Rogers
8 G. Pelizzi	22 V. Zabel	35 L. Riva
9 G. Evrard	23 W.A. Wooster	36 N. Faraboli
10 G. Fava	24 H. Hauptman	37 A. Krajewski
11 E.M. van den Hark	25 A.L. Spek	38 A. Musatti
12 H. Krabbendam	26 D. Zobel	39 G. Gilli
13 F. Lazarini	27 D. Viterbo	40 P. Sgarabotto
14 L. Battaglia		

Fig. 2b: *Legend of Participants at the Direct Methods in X-ray Crystallography 1973, Lectures by H. Hauptman.*

Enschede Crystallographic Computing School, The Netherlands, 24 July - 1 August 1978

The central theme of this pre-IUCr Congress computing school was around software packages. Computing schools at that time had massive attendance. The weather was excellent. The school was organised on the campus of the Technical University Enschede, near the German border. Many will remember the nice weather, the cycling excursion and the open air conference dinner.

Fig. 3a: *Participants at the Enschede Crystallographic Computing School, 1978. Legend listing participants in Fig 3b overleaf*

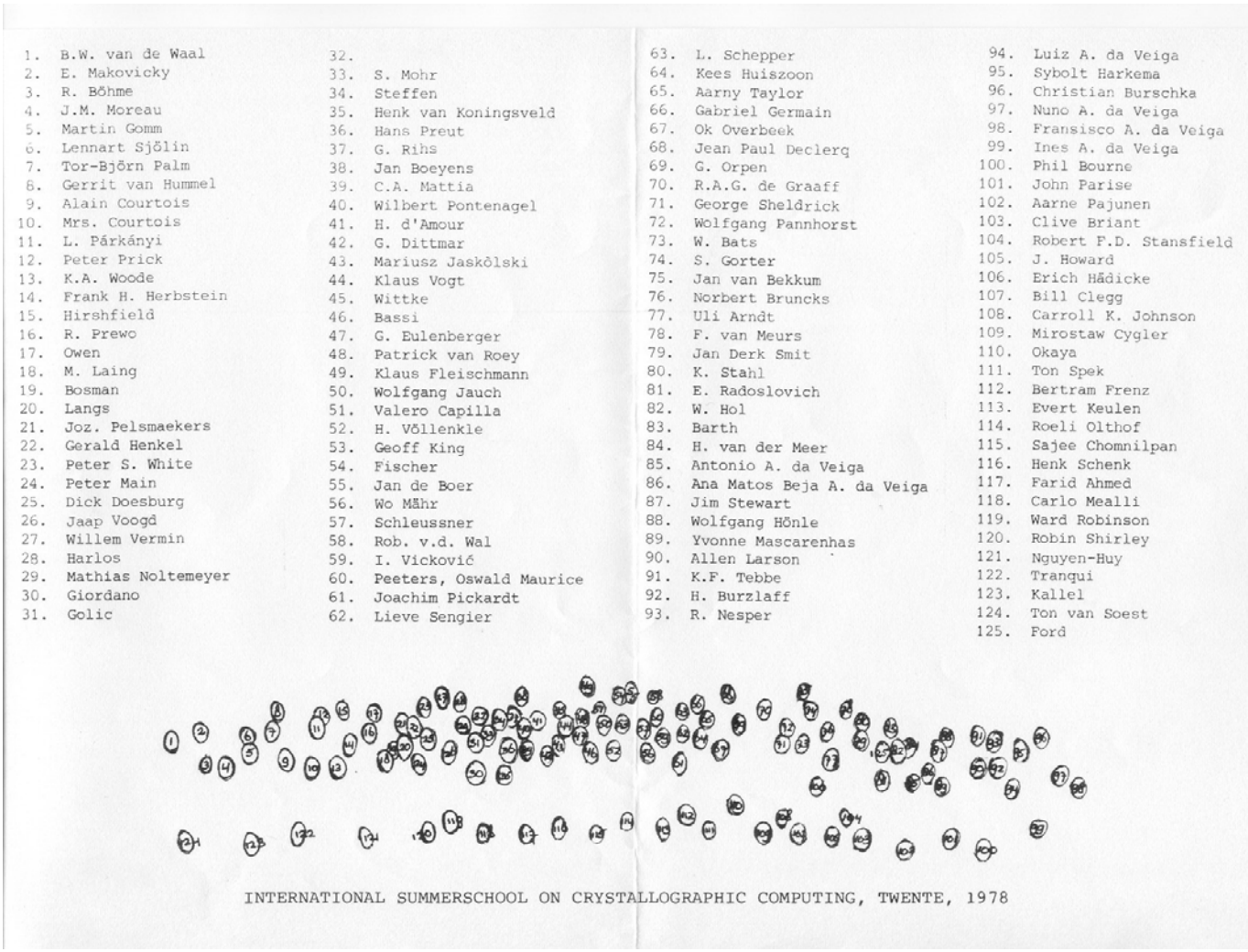


Fig. 3b: *Legend listing participants at the Enschede Crystallographic Computing School, 1978.*

Photographs from the Erice 1978 School on Direct Methods for Solving Crystal Structures (5th Course: 27 March to 9 April 1978) Director : Giuseppe Allegra, Milan

Lodovico Riva
Dip. di Scienze della Terra e Geologico-Ambientali, Università di Bologna, Piazza di Porta S. Donato 1, I-40126, Bologna. E-mail: lodovico.riva@unibo.it : http://www.crystalerice.org/Pastactivity/1978/1978.htm

Due to colds and coughings gathered during previous Erice meetings, to celebrate the fifth crystallographic event participants were donated a wool scarf made in Erice, as a souvenir; in order to encourage them to wear the scarf, the organizers had sticked sheets at the walls in Erice with a warning "Crystallographers, protect your neck!" Soon after the meeting, the biophysicists came for their school and watching those sheets still hanging at the walls, were asking themselves what sort of dangerous life crystallographers had been experiencing few days earlier.

Fig. 1a: *Participants at the Erice 1978 School on Direct Methods for Solving Crystal Structures. Legend listing participants in Fig 1b overleaf*

I G. Polidori
2 W. Vermin
3 W. Paton
4 G. Olthof
5 O. Overbeek
6 A. Nunzi
7 R. Spagna
8 A. Mugnoli
9
IO G. Chiari
II S. Bruckner
I2 L. Kutschabsky
I3 R. Norrestam
I4 C. Briant
I5 M. Vlasse
I6 V. James
I7 E. Paulus
I8 R. Gunawardane
I9 S. Narasinga Rao
20 K. Elchhorn
2I A. Terzis
22
23 R. Stansfield
24
25 V. Rivera
26 L. Smart
27 K. Sheldrik
28 G. Sheldrik
29 G. Ferguson
30 E. Ljungstrom
3I M. Rojas
32 K. Tomita
33 R. Schaeffer
34 H. Schenk
35 G. Germain
36 R. Roques
37 G. Punte
38 F. Mo
39 O. Jarchow
40 D. Watkin
4I K. Woode
42 S. Toure
43 P. Spadon
44 J. Rodgers
45 J. Trotter
46 L. Niinisto
47 D. Viterbo
48 B. Mehrotra
49 W. Winter
50 S. G. Biswas
5I G. Rigotti
52 T. Skarzynski
53 E. Wajsman
54 F. Mazza
55 Pinola
56 Lodovico
57 H. Kuppers
58 E. Duée
59 Z. Dauter
60 Z. Lipkowska
6I I. Karle
62 E. Arzi
63 P. Jones
64 M. le Bars
65 I. Vickovic
66 W. Ficher
67 A. L. Spek
68 V. Pavone
69 B. Sanni
70 N. van der Putten
7I E. Holt
72 K. Simon
73 G. Tsoucaris
74 P. Main
75 H. Krabbendam
76 P. Rogl
77 U. Shmueli
78 C. Giacovazzo
79 H. Hauptman
80 H. Boler
8I M. Foulon
82 G. Allegra
83 E. Hadicke
84 V. I. Simonov
85 I. Goldberg
86 A. Thozet
87 M. M. Woolfson
88 I. Kjoller Larsen
89 D. Sayre
90 N. Fiagbe
9I B. Stensland
92 P. Beursken
93 J. Karle
94 K. Hulm
95 J. Daly

Fig. 1b: *Legend listing participants at the Erice 1978 School on Direct Methods for Solving Crystal Structures. Courtesy of Ton Spek.*

Fig. 2: *Jerome Karle at the Erice 1978 School on Direct Methods for Solving Crystal Structures.*

Lectures and Lecturers

- Probability Distribution of Structure Factors G. Allegra, Polytechnic, Milan,
- Origin Fixing G.T. De Titta, Medical Research Foundation, Buffalo, USA
- Symbolic Addition C. Giacovazzo, Università di Bari, I
- Multisolution Methods H. Hauptman, Medical Research Foundation, Buffalo, USA
- Phase Refinement and Extension Techniques I.L. Karle, Naval Research Laboratory, Washington, USA
- Multiple Phase Relationships J. Karle, Naval Research Laboratory, Washington, USA
- Theory of Cosine Invariants P. Main, University of York, UK
- Theory of Inequalities R. Norrestam, Technical University Lingby, DK
- Relationships between Relationships D. Sayre, IBM Research Laboratories, Yorktown Heights, USA
- Matrix Methods H. Schenk, University of Amsterdam, NL
- The Theory of Inequalities G.M. Sheldrick, University of Cambridge, UK
- Use of Chemical Information D. Viterbo, University of Turin, I
- Magic Integers M.M. Woolfson, University of York, UK
- Application to Macromolecules

The above has been extracted from the announcement printed by the Majorana Centre months before the meeting started. The actual contributions had often different titles and are "buried" into three volumes of collected lecture notes as witnessed by the photo below.

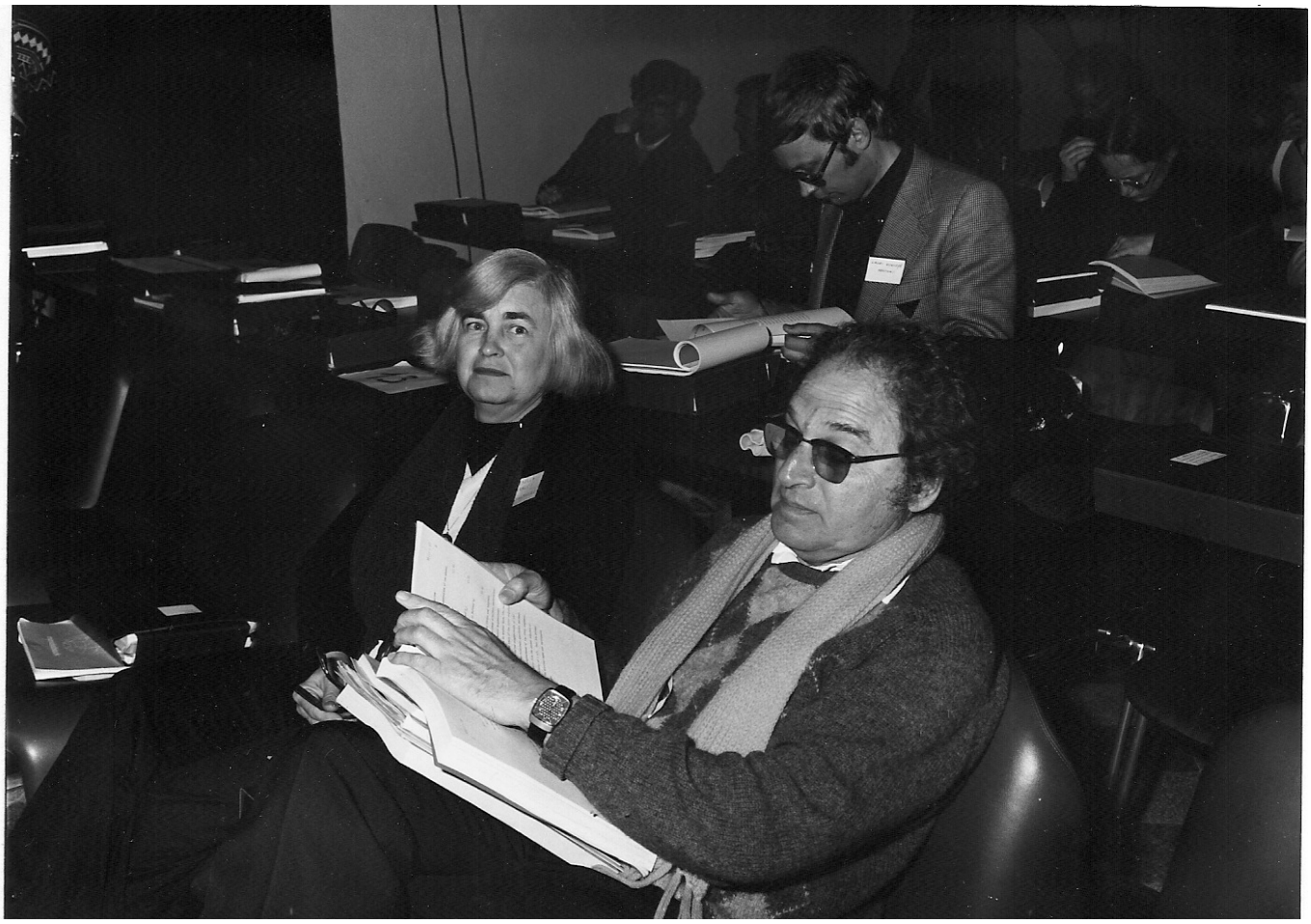


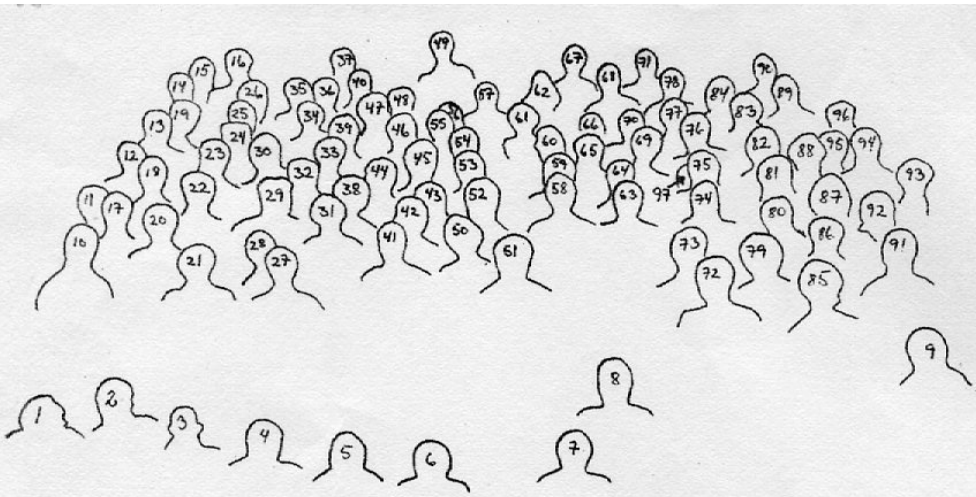
Fig. 3: *Isabella L. Karle and Herbert A. Hauptman at the Erice School: Direct Methods for Solving Crystal Structures, 27 March to 9 April 1978.*

Group photograph from the NATO School on the Experimental Aspects of X-ray and Neutron Diffraction, Aarhus, Denmark, 1972

David J. Watkin
Chemical Crystallography, Chemistry Research Laboratory, Oxford, OX1 3PD, United Kingdom. E-mail: david.watkin@chem.ox.ac.uk

The 1960s and 1970s were the heyday of crystallographic schools, with many devoted to Computing and to Direct Methods. The proceedings of many of the Computing Schools were published, and the earlier ones are still a useful source of background information. The NATO Advanced Institute in Aarhus was the only international meeting I can recall which was dedicated to broad-based experimental diffraction techniques, though there has been a continuing succession of biennial Neutron Schools in Oxford. At this Study Institute I was lucky enough to hear Ewald and Walter Hamilton speak and to head Don Sands warn about the hazards of trying to compute the average of an apple and an orange. Bruce Forsythe demonstrated something using a video recorder. Of the non-crystallographic activities I remember the visit to the Old Town, attending a performance of the L'incoronazione *di Poppea* (in Danish) and seeing Tollund Man, an ancient sacrificial victim preserved in a peat bogs.

Fig. 1a: *Participants at the NATO Advanced Institute on the Experimental Aspects of X-ray and Neutron Diffraction, Aarhus University, Aarhus, Denmark, 31 July-11 August, 1972. Legend listing participants in Fig 1b overleaf*



1 H. Arnold
2 Hans Bartl
3 Jim Speakman
4 Stuart Pawley
5 J. Amigo
6 Svend Erik Rasmussen
7 Charlotte Kallesøe
8 J. de Boer
9 Kirsten Petersen
10 K.F. Fischer
11 W. Schäfer
12 A. Faruqui
13 Camden Hubbard
14 E. Prince
15 Britta Lundtoft
16 Olliver Lindqvist
17 Dirk Feil
18 Fred Ross
19 B. Deppisch
20 Axel Nørlund Christensen
21 Kirsten Streib
22 Robert Thomas
23 Uli Arndt
24 K. Tebbe
25 Karl Peder Larsen
26 Lars Kryger
27 Bill Streib
28 Helen Berman
29 Hans Leopold
30 Walter Graeff
31 Raul Mariezcurrena
32 Peter Seidel
33 A. Gieren
34 Uri Shmueli
35 Jens Nyborg
36 John Metcalf
37 Alex Mehlsen-Sørensen
38 Frank Pedersen
39 L. Leiserowich
40 Jacob Danielsen
41 Niels Thorup
42 H. Mez
43 B. Faught
44 L. Golic
45 John Reid
46 Louis Delbaere
47 Asbjørn Hordvik
48 Ray Young
49 Finn Krebs Larsen
50 Alan Hazell
51 Paul Beurskens
52 Claes Stålhandske
53 David Watkin
54 Sax Mason
55 M. Colapietro
56 A. Albinati
57 Nomy Ponce
58 H. Rietveld
59 Fred Kundell
60 M. Stringfellow
61 Rita Hazell
62 Marianne Mortensen
63 Sofokles Filippakis
64 David Gilmore
65 Karen Kaas
66 J. Lamotte
67 M. Agostinelli
68 M. Webster
69 M. Bonnet
70 F. A. Schröder
71 P. Sgarabotto
72 Philip Coppens
73 Henk Schenk
74 A. Duisenberg
75 K. Fleischman
76 P. Becker
77 Bernt Klewe
78 Stig Aleby
79 Richard Thornley
80 Arne Andresen
81 Jim Edmonds
82 Walter Hamilton
83 Bruno Lundberg
84 Rolf Norrestam
85 Richard Nelmes
86 Bruce Forsyth
87 F. Scarbrough
88 Tom Koetzle
89 Stig Åsbrink
90 Galen Lenhert
91 Keith Rouse
92 R. de Graaff
93 Terry Willis
94 Ivar Olovsson
95 Jorun Sletten
96 Jan-Olof Lundgren
97 Sine Larsen

Fig. 1b: *Legend listing participants at the NATO Advanced Institute on the Experimental Aspects of X-ray and Neutron Diffraction, Aarhus University, Aarhus, Denmark, 31 July-11 August, 1972.*

Call for Contributions to the Next CompComm Newsletter

The next issue of the Compcomm Newsletter is expected to appear around October of 2010 with the primary theme of Age Concern relevant to Protein Crystallography and/or Powder Diffraction. 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 (CNBC),
National Research Council of Canada (NRC),
Building 459, Station 18, Chalk River Laboratories,
Chalk River, Ontario, Canada, K0J 1J0
Tel: (613) 584-8811 ext: 43719
Fax: (613) 584-4040
E-mail: lachlan.cranswick@nrc.gc.ca