IUCr Computing Commission

Commission on Crystallographic Computing

International Union of Crystallography
http://www.iucr.org/iucr-top/comm/ccom/

Newsletter No. 3, January 2004
http://www.iucr.org/iucr-top/comm/ccom/newsletters/

Table of Contents

(This Issue's Editor: Lachlan Cranswick)

(Editor's warning – unless you want to kill 70 pages worth of forest – DO NOT press the "print" button. For hardcopies – you may like to only print out the articles of personal interest.)

CompComm Chairman’s Message

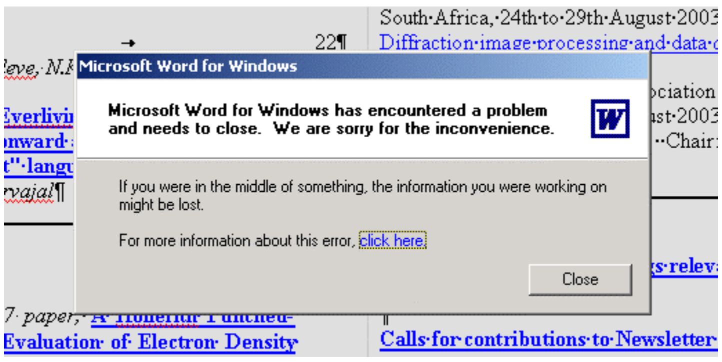
The editor has managed again to collect many interesting contributions for our newsletter. For everybody, there is something to their liking. As an old-fashioned Fortran programmer, it is pleasing to read the article 'Once and Everliving FORTRAN' by Juan Rodríguez-Carvajal that provides some rational for my addiction. In contrast there is the article by Grosse-Kunstleve et al. on the modern toolbox approach of software development. Again there is an historical article, this time about the dark ages of crystallographic computing on the use of Hollerith Punched Cards as a computation tool; for the calculation of Fourier maps. Compared to modern methods which take milliseconds or less nowadays, this was an enormous task at the time.

Finally, there is an announcement and information about the next crystallographic computing school, to be held prior to the Florence IUCr2005 congress at an excellent location; an ex-14th century monastery now run as a guest house and conference centre by the University of Siena, in Tuscany, Italy.

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

From the Editor of Newsletter No. 3

As is the tradition in this section, the editor was going to waffle on something lyrical; but then Microsoft Word 2000 decided that while opening and editing this file was OK, any attempt to save was deserving of the equivalent of a Blue-Screen-Of-Death (BSOD) (as shown below). This is not what someone on the final edits of a newsletter wants to see. An **Edit Copy**, followed by an **Edit Paste** into a new document seems to have recovered most things, except some strange random font resizing, which have hopefully all been found. Open source enthusiasts will no doubt draw many relevant and vocal conclusions from all of this.



Microsoft Word for Windows

Microsoft Word for Windows has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

For more information about this error, click here.

Nervously expecting this Microsoft Word software to self destruct at any moment, I would like to quickly thank all the authors and symposium reviewers for their contributions. Also, anyone with some artistic talent is requested to submit something for the logo competition.

On the historical side, and as mentioned in the last edition, this issue includes a reprint on one of the first applications of computers to crystallography, the 1947 paper, *“A Hollerith Punched-card Method for Evaluation of Electron Density in Crystal Structure Analysis”* by E.G. Cox, L. Gross and G.A. Jeffrey, Proc. Leeds Phil. Soc., 5, 1-13, (1947). This includes a modern introduction by Professor Stanley C. Nyburg, who was a user of the method back in the 1940’s. It is hoped for the next edition to have an invited article on the development of crystallographic computing software in the late 1950’s to early 1960’s. Other perspectives, including suggestions of titles to reprint, in the history of crystallographic computing are very welcome.

Lachlan Cranswick (Lachlan.cranswick@nrc.gc.ca)

THE IUCR COMMISSION ON CRYSTALLOGRAPHIC COMPUTING - TRIENNIUM 2003-2005

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

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

Lachlan M. D. Cranswick
Neutron Program for Materials Research (NPMR),
National Research Council (NRC),
Building 459, Station 18, Chalk River Laboratories,
Chalk River, Ontario, Canada, K0J 1J0
Tel: (613) 584-8811 ext: 3719
Fax: (613) 584-4040
E-mail: lachlan.cranswick@nrc.gc.ca
WWW: http://lachlan.bluehaze.com.au/

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

Dr Ralf Grosse-Kunstleve
Lawrence Berkeley Lab
1 Cyclotron Road,
BLDG 4R0230,
Berkeley, CA 94720-8235, USA.
Tel: 510-486-5713
Fax: 510-486-5909
E-mail: rwgk@yahoo.com
WWW: http://cci.lbl.gov/

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

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

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

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

Consultants

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

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

Siena-2005 - IUCr CompComm Crystallographic Computing School

Certosa di Pontignano, Siena, Tuscany, Italy. 18th to 23rd August, 2005

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

Fig 1: an artist's rendition of the Certosa di Pontignano. With its origins in the 1300's as a monastery, it is now run by the University of Siena, and is the venue for the Siena 2005 Computing School.

Computing forms a major part of most crystallographic work. In the past, many crystallographers wrote their own programs (or adapted existing code) for their particular application. Some of those programs (e.g. ORTEP, SHELX) turned out to be useful on a wider scale than for just the local research and available in nearly every smallmolecule lab, others have found a place in the software museum of LeBail.

The current trend is the use of commercial software packages. On the positive side that can mean more stability, responsibility and continuity. On the negative side it can mean undisclosed proprietary algorithms, black box operation and slow adoption of new developments. There is some tendency among the current generation of crystallographers that when there is no button available on the program menu for a desired feature it just cannot be done. In the past one would write his own program to do the job.

Fig 2: Photographs of the Certosa di Pontignano, the venue for the Siena 2005 Computing School.

Computing schools have played in the past an important role in making crystallographic computing efforts known on a wider scale. An excellent example is the 1969 school where a lot of least-squares and absorption correction code was presented in considerable detail and still in use to the present day.

Most of the people that attended this 1969 school are now retired or close to retirement. With the computing school to be held in Siena prior to the Florence IUCr2005 congress we hope to provide a similar experience for the next generation of software developers.

Ton Spek

Chairman or the IUCr Computing Commission, Utrecht University, the Netherlands.

E-mail: a.l.spek@chem.uu.nl ; WWW: http://www.cryst.chem.uu.nl/spea.html

A preliminary announcement of the IUCr Commission on Crystallographic Computing

Siena 2005 - Crystallographic Computing School

Certosa di Pontignano,
University of Siena, Italy
18th - 23rd August 2005
(just prior to the Florence IUCr 2005 congress)

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

School Organisers: Prof Anthony Spek (Utrecht), Prof. Marcello Mellini (Siena), Prof. Alessandro Gualtieri (Modena), Dr Harry Powell (Cambridge), Lachlan Cranswick (NRC Chalk River)
Consultants: Dr David Watkin (Oxford), Dr Simon Parsons (Edinburgh)

Siena, Tuscany
ROME
http://www.cia.gov/cia/publications/factbook/

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

"principles & methods"
"joining things together"
"crystallographic implementations"
"selected topics in crystallography"
"special methods"

The City

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

The Venue

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

School Aims

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

Università degli Studi di Siena

Reading Binary Data

Scott A. Belmonte,
91 Lord Nelson Street, Warrington, Cheshire, WA1 2JF, U.K. E-mail: scott.belmonte@ntlworld.com - WWW: http://www.ccp14.ac.uk/ccp/web-mirrors/scott-belmonte-software/

Abstract

This article describes how to read binary data in C and FORTRAN. Knowing how to read binary data gives a scientific programmer the opportunity to write software to analyse data collected by proprietry scientific instruments. This enables the programmer to customise the analysis process and to be freed from the restrictions of the software supplied by the company that manufactured the scientific instrument (for example, a CCD diffractometer).

Introduction

All files on a modern computer are binary files that can be though of as streams of 8-bit bytes. The meaning of the bytes depends on the context in which they are being used. For example, the bytes of an ASCII encoded text file represent entries in the ASCII character set. Most programming languages have built-in routines for handling ASCII encoded text files. This is why a distinction is often made between ASCII and binary. However, in reality, text files are binary files.

The key to reading a binary file is knowing what the bytes of the file represent. Once this is known, it is a simple job to convert the bytes into the approriate internal representation of the programming language being used. However, finding out the file format is often the hardest part. The next section gives some tips on how to decode unknown file formats.

In the following sections, examples will be given of how to read and convert binary data in C and FORTRAN. Each section heading contains the programming language that is used for the examples in that section. If the reader is not interested in that particular language then the section can be skipped. Examples of how to use the code will also be given.

File Formats

Knowing the format of the file to be read is the biggest step towards extracting data from it. The best place to find this information is from the manufacturer of the software that created the file; possibly from the manuals that came with the software.

However, sometimes manufacturers are reluctant to give out such information. In this case, it might still be possible to decode the file format using a hex dump utility and some prior information as to what the file contains. For example, if you know that a file contains a 512x512 pixel CCD image then it is likely that the image data are in a contiguous block at the end of the file with one, two or four bytes per pixel. This gives a file size of 262144, 524288 or 1048576 bytes plus some for header information. If the file size is 1052676 bytes, for example, then it is likely that the file has a 4100 byte header followed by 512x512 4-byte (32-bit) pixels starting at a byte-offset of 4100 in the file. By guessing this, it is possible to use the routines below to read the image data.

A hex dump utility is also a very useful tool for determining file formats. The output from such a utility will look something like the following:

```
0       : 49 49 2A 00 B0 05 20 00      II*... .
8       : 6C 01 5D 01 85 01 F2 02      l.].....
16      : BA 02 92 02 75 02 6D 02      ....u.m.
24      : 6D 02 70 02 59 02 55 02      m.p.Y.U.
32      : 4D 02 47 02 42 02 30 02      M.G.B.0.
40      : 22 02 1D 02 1B 02 0F 02      ".......
48      : 04 02 06 02 FE 01 FA 01      ........
56      : F6 01 01 02 ED 01 EB 01      ........
64      : E4 01 E0 01 D5 01 E7 01      ........
```

The example above shows the first 72 bytes of a TIFF file.

Often patterns can be deduced from the hex dump that can be used to determine at least parts of the file format.

Once the file format is know or deduced then the file can be read into memory and the data converted for analysis. The following sections detail how to do this in FORTRAN and C.

Endian (C/FORTRAN)

Endian is a property of processors. Basically, it is how a processor orders the bytes of the multibyte sequence. A processor can be big-endian or little-endian. This is an important property to take into account when reading binary files.

In natural languages, numbers are written from left to right from the most significant digit to the least significant. Processors can order the numbers either from the most significant byte to the least significant (big-endian) or from the least significant byte to the most significant (little-endian). The two-byte hexidecimal number ABCD would be written to a binary file as follows:

Big-endian:	AB CD
Little-endian:	CD AB

The four-byte hexidecimal number 1234ABCD would be written as follows:

Big-endian:	12 34 AB CD
Little-endian:	CD AB 34 12

If a binary file was written by a machine that has a different endian to the machine reading the binary file then the bytes of multibyte numbers need to be swapped before being converted to their internal representation.

It is not always possible to determine the endian (byte-ordering) of a file. Some file types specify the endian by using specific byte markers near the beginning of the file. The TIFF file type is an example of such a file type. Other file types contain no endian information so the endian needs to be guessed; usually it is the same as the endian of machine that wrote the file. For informational purposes, Intel processors are generally little-endian and Sun SPARC processors are big-endian.

In FORTRAN, the endian of the processor can be determined with the following routine:

```
C
C**********************************************************************
C     Routine: BIGEND_CPU
C
C     Description:
C        Returns .TRUE. if the machine is big endian
C**********************************************************************
      LOGICAL FUNCTION BIGEND_CPU()
      IMPLICIT NONE
C
C     Variables
C
      BYTE         B(2)
      INTEGER*2    I2
      EQUIVALENCE (B, I2)
C
      I2 = 1
      BIGEND_CPU = (B(1) .NE. 1)
      RETURN
      END
C
```

A slightly non-equivalent C version is (split into two files ‘readbin.h’ and ‘readbin.c’ as indicated):

```
/******
 * readbin.h
 */
typedef enum
{
    MACHINE_LITTLE_ENDIAN,
    MACHINE_BIG_ENDIAN
} e_endian;

e_endian get_endian(void);

/******
 * readbin.c
 */

#include “readbin.h”

/**********************************************************************
 *    Function: get_endian
 *
 *    Description:
 *       Returns MACHINE_BIG_ENDIAN if the machine is big endian.
 *       Returns MACHINE_LITTLE_ENDIAN if the machine is little endian.
 *********************************************************************/
e_endian get_endian(void)
{
    unsigned char *ch;
    int i = 1;

    ch = (unsigned char *) &i;

    if (*ch != 1)
    {
        return MACHINE_BIG_ENDIAN;
    }
    else
    {
        return MACHINE_LITTLE_ENDIAN;
    }
}
```

Opening Binary Files (FORTRAN)

Routines to read binary files in FORTRAN cannot be entirely portable between operating systems and/or compilers because FORTRAN reads files in records and the definition of a record depends on the compiler or operating system. Code that compiles and operates correctly on one machine might need to be modified to get it to work on another. However, the changes required are normally small. Potentially non-portable code in the examples will be pointed out.

The following code is contained in a FORTRAN include file called ‘readbin.inc’, which is included by most routines used to read binary files:

```
C
      INTEGER REC_SIZE  ! The file record size in bytes
      INTEGER UNT       ! The unit number that file is opened on
      PARAMETER (REC_SIZE = 2048)
      PARAMETER (UNT = 11)
C
      BYTE    BUFFER(REC_SIZE) ! Read buffer for records
      INTEGER LAST_REC         ! The last record read into BUFFER
      LOGICAL BIG_END          ! .TRUE. if the file is big endian
      LOGICAL SWAP             ! If .TRUE. the bytes need to be
C                              ! before they can be used
      COMMON /BIN_FILE/ BUFFER, LAST_REC, BIG_END, SWAP
      SAVE /BIN_FILE/
C
```

Files will be read in chunks of 2048 bytes (REC_SIZE) stored in the byte array BUFFER. The record size is arbitrary. However, there is a trade-off between memory and speed to consider. More memory is required for larger record sizes but fewer disc accesses (which are inherently slow compared to memory accesses) are required when reading a complete file.

The LAST_REC integer contains the number of the last record read into BUFFER. LAST_REC is used to determine whether the requested data have already been read into BUFFER; the data will be read directly from the buffer rather than from the disc again if this is the case.

BIG_END is TRUE is the file being read is big-endian as explained in the 'Endian' section above. This can be hard-coded, if the endian of the file is known, or determined from the file, if the file type contains endian information. The SWAP variable can be calculated from the value of BIG_END and the result of the BIGEND_CPU routine given in the 'Endian' section above:

```
C
C     The following variable, BIG_END, is .TRUE. if the data
C     in the binary file is big endian. This can either be hard
C     coded, as it is here, if the endian of the file is known,
C     or it can be determined from the file itself if the file
C     provides such information.
C
      BIG_END = .FALSE.
C
C     If the machine has a different endian than the file
C     then SWAP is set to .TRUE.
C
      SWAP = (BIGEND_CPU() .XOR. BIG_END)
C
```

The following routines in this section and the next should be placed in a single FORTRAN file (*e.g.* 'readbin.f').

```
C
C**********************************************************************
C     Routine: OPEN_BIN_FILE
C
C     Description:
C        Opens a binary file whose name is stored in NAME.
C
C        NB. Uses a fixed unit number. This means only one file
C        can be open at a time. CLOSE_BIN_FILE must be called
C        before OPEN_BIN_FILE can be called again.
C
C
C        Returns: 0 file successfully opened,
C                 1 error opening file
C**********************************************************************
      INTEGER FUNCTION OPEN_BIN_FILE(NAME)
      IMPLICIT NONE
      INCLUDE 'readbin.inc'
C
C     Parameters
C
      CHARACTER*(*) NAME        ! File name
C
C     Functions
C
      LOGICAL BIGEND_CPU
C
C     Variables
C
      LOGICAL   OPEN            ! .TRUE. if a file is already open
C
C     Initialise the LAST_REC common. 0 means that BUFFER contains
C     no valid record.
C
      LAST_REC = 0
C
C     Check that a file isn't currently open. These routines
C     can only work with one file at a time.
C
      INQUIRE(UNIT=UNT, OPENED=OPEN)
      IF (OPEN) THEN
         WRITE(*,*) '** OPEN_BIN_FILE error: File already opened.'
         WRITE(*,*) '** Only one file can be opened at a time.'
         GOTO 901
      ENDIF
C
C     Open file
```

```
C
      OPEN(UNIT=UNT, FILE=NAME, STATUS='OLD', FORM='UNFORMATTED',
     $     ACCESS='DIRECT', RECL=REC_SIZE, ERR=901)
      REWIND(UNIT=UNT)
C
      OPEN_BIN_FILE = 0
      RETURN
C
C     Error traps
C
 901  OPEN_BIN_FILE = 1        ! Error opening file
      RETURN
      END
C
```

Valid record numbers start from 1 so setting LAST_REC to 0 indicates that BUFFER does not contain valid data.

Since there is only one buffer available, it is only possible to read one file at a time or else the buffer could become corrupted. A check is made to make sure that a binary file is not already open.

The OPEN statement is a possible source of non-portability. If the compiled program works correctly on one computer but seems to fail on another platform then it is possible that the RECL option of the OPEN statement is being interpreted differently between platforms. There are normally two options: RECL is the record length in bytes, or RECL is the record length in (4- or 8-byte) words. If the open statement above does not seem to work try:

```
      OPEN(UNIT=UNT, FILE=NAME, STATUS='OLD', FORM='UNFORMATTED',
     $     ACCESS='DIRECT', RECL=REC_SIZE/4, ERR=901)
```

or:

```
      OPEN(UNIT=UNT, FILE=NAME, STATUS='OLD', FORM='UNFORMATTED',
     $     ACCESS='DIRECT', RECL=REC_SIZE/8, ERR=901)
```

The close file routine is quite simple:

```
C
C**********************************************************************
C     Routine: CLOSE_BIN_FILE
C
C     Description:
C        Closes the binary file.
C**********************************************************************
      SUBROUTINE CLOSE_BIN_FILE()
      IMPLICIT NONE
      INCLUDE 'readbin.inc'
C
      CLOSE(UNIT=UNT)
      END
C
```

Reading Binary Files (FORTRAN)

The guts of the FORTRAN binary file reader are contained in the following routines (READ_BIN_RECORD is a utility function used by READ_BIN_FILE).

```
C
C**********************************************************************
C     Routine: READ_BIN_FILE
C
C     Description:
C        Reads bytes START_BYTE to END_BYTE, inclusive, from a binary
C        file into the array DATA.
C        NB. START_BYTE and END_BYTE are zero offset.
C
C        Returns 0 if successful, 1 otherwise
C**********************************************************************
      INTEGER FUNCTION READ_BIN_FILE(DATA, START_BYTE, END_BYTE)
      IMPLICIT NONE
      INCLUDE 'readbin.inc'
C
C     Parameters
C
      BYTE    DATA(*)            ! The array to read the bytes into
      INTEGER START_BYTE         ! The byte to start reading from
      INTEGER END_BYTE           ! The byte to finish reading at.
C
C     Functions
C
      INTEGER READ_BIN_RECORD
C
C     Variables
C
      INTEGER START_REC          ! Start record
      INTEGER END_REC            ! End record
      INTEGER TOTAL_BYTES        ! The total number of bytes to read
      INTEGER NEXT_BYTE          ! The next byte to write into DATA
      INTEGER CUR_REC            ! Current record being read
      INTEGER SKIP_BYTES         ! Num bytes to skip in first record
      INTEGER NUM_BYTES          ! Num bytes to read from first record
      INTEGER STATUS             ! Status of the read
      INTEGER I                  ! Loop counter
C
C     Check for errors in the parameters START_BYTE and END_BYTE.
C
      IF ((START_BYTE .GT. END_BYTE) .OR. (START_BYTE .LT. 0)) THEN
         WRITE(*,*) '** READ_BIN_FILE Error: Bad START_BYTE or
     $        END_BYTE'
         WRITE(*,*) '** START = ', START_BYTE, ', END = ', END_BYTE
         GOTO 911
      ENDIF
C
C     Calculate the first and last records that have to be read, and
C     the total number of bytes that have to be read.
C
      START_REC   = 1 + START_BYTE/REC_SIZE
      END_REC     = 1 + END_BYTE/REC_SIZE
      TOTAL_BYTES = END_BYTE - START_BYTE + 1
C
      NEXT_BYTE = 1
      CUR_REC   = START_REC
C
C     Read the first record (unless it already is in BUFFER).
C
      STATUS = 0
      IF(CUR_REC .NE. LAST_REC) THEN
         READ(UNIT=UNT, REC=CUR_REC, IOSTAT=STATUS) BUFFER
      ENDIF
C
C     Copy the required bytes from BUFFER to DATA.
C
      SKIP_BYTES = START_BYTE - (START_REC - 1)*REC_SIZE
      NUM_BYTES  = MIN(REC_SIZE - SKIP_BYTES, TOTAL_BYTES)
      DO I = 1, NUM_BYTES
         DATA(I) = BUFFER(I + SKIP_BYTES)
      ENDDO
      NEXT_BYTE = NEXT_BYTE + NUM_BYTES
      IF(STATUS .NE. 0) GOTO 911
C
C     Read anymore whole records directly into DATA.
C
      DO CUR_REC = START_REC + 1, END_REC - 1
         STATUS = READ_BIN_RECORD(DATA(NEXT_BYTE), CUR_REC)
         IF(STATUS .NE. 0) GOTO 911
         NEXT_BYTE = NEXT_BYTE + REC_SIZE
      ENDDO
C
C     If needed, read the last record into BUFFER, then transfer to DATA.
C
      CUR_REC = END_REC
      IF (CUR_REC .NE. START_REC) THEN
         READ(UNIT=UNT, REC=CUR_REC, IOSTAT=STATUS) BUFFER
         DO I = NEXT_BYTE, TOTAL_BYTES
            DATA(I) = BUFFER(I - NEXT_BYTE + 1)
         ENDDO
      ENDIF
C
C     Store the number of the record that BUFFER contains, then return.
C
      LAST_REC = CUR_REC
      READ_BIN_FILE = 0
      RETURN
C
C     Error occured. Set LAST_REC = 0 so we don't reuse BUFFER contents.
C
C     Ignore errors that occur when reading END_REC since this
C     might be the last record in the file and may cause an error
C     because it is less than REC_SIZE in size.
C
 911  IF (CUR_REC .EQ. END_REC) THEN
         LAST_REC = CUR_REC
         READ_BIN_FILE = 0
      ELSE
         LAST_REC = 0
         READ_BIN_FILE = 1
      ENDIF
      RETURN
C
      END
C
C
C
C**********************************************************************
C     Routine: READ_BIN_RECORD
C
C     Description:
C        Reads the record RECORD for the open file on unit UNT into
C        ARRAY.
C
C        Returns 0 if successful, 1 otherwise
C**********************************************************************
      INTEGER FUNCTION READ_BIN_RECORD(ARRAY, RECORD)
      IMPLICIT NONE
      INCLUDE 'readbin.inc'
C
C     Parameters
C
      BYTE    ARRAY(REC_SIZE)
      INTEGER RECORD
C
C     Variables
C
      INTEGER STATUS
C
      READ(UNIT=UNT, REC=RECORD, IOSTAT=STATUS) ARRAY
      READ_BIN_RECORD = STATUS
      RETURN
      END
C
```

READ_BIN_FILE reads the file in records. It calculates what records START_BYTE and END_BYTE lie in (START_REC and END_REC, respectively). If the record that START_BYTE is in is already in BUFFER then the data are copied directly from BUFFER into the raw data array (DATA). This speeds up multiple accesses to the same record by preventing multiple disc reads.

If there are one or more whole records between START_REC and END_REC (*i.e.* if END_REC – START_REC is greater than one) then the records are read from disc directly into the appropriate part of DATA using the utility function READ_BIN_RECORD.

Finally, END_REC is read into BUFFER and the bytes up to END_BYTE are copied into DATA. If there have been no errors, LAST_REC is set to END_REC to indicate that BUFFER now contains the END_REC record's data.

READ_BIN_FILE can be used to read arbitrary chunks of the open file into memory. The data in memory are the raw data as read from disc. The raw data need to be converted into meaningful values according to the file format, and multibyte sequences need to be swapped if necessary.

Bytes can easily be converted into characters using the CHAR built-in function. For example:

```
C
C     Assume a file has already been successfully opened
C
      CHARACTER CH
      BYTE      RAWDATA(2)
C
C     Read first two bytes of file and convert the first byte to a
C     character.
C
      IF (READ_BIN_FILE(RAWDATA, 0, 1) .EQ. 0) THEN
          CH = CHAR(RAWDATA(1))
      ENDIF
C
```

2-byte (16-bit) and 4-byte (32-bit) integers can be converted using the following routines. These routines swap the bytes, if appropriate, before they are converted.

```
C
C**********************************************************************
C     Routine: READ_BIN_SHORT
C
C     Description:
C        Converts the 2 bytes in RAW to INTEGER swapping
C        the bytes if necessary.
C**********************************************************************
      INTEGER FUNCTION READ_BIN_SHORT(RAW)
      IMPLICIT NONE
      INCLUDE 'readbin.inc'
C
C     Parameters
C
      BYTE         RAW(2)
C
C     Variables
C
      BYTE         B(2)
      INTEGER*2    I2
      EQUIVALENCE (B, I2)
C
      IF (SWAP) THEN
         B(1) = RAW(2)
         B(2) = RAW(1)
      ELSE
         B(1) = RAW(1)
         B(2) = RAW(2)
      ENDIF
C
      READ_BIN_SHORT = I2
      RETURN
      END
C
C
C
C**********************************************************************
C     Routine: READ_BIN_LONG
C
C     Description:
C        Converts the 4 bytes in RAW to INTEGER swapping
C        the bytes if necessary.
C**********************************************************************
      INTEGER FUNCTION READ_BIN_LONG(RAW)
      IMPLICIT NONE
      INCLUDE 'readbin.inc'
C
C     Parameters
C
      BYTE         RAW(4)
C
C     Variables
C
      BYTE         B(4)
      INTEGER*4    I4
      EQUIVALENCE (B, I4)
C
      IF (SWAP) THEN
         B(1) = RAW(4)
         B(2) = RAW(3)
         B(3) = RAW(2)
         B(4) = RAW(1)
      ELSE
         B(1) = RAW(1)
         B(2) = RAW(2)
         B(3) = RAW(3)
         B(4) = RAW(4)
      ENDIF
C
      READ_BIN_LONG = I4
      RETURN
      END
C
```

Floating point numbers can be read in a similar manner to integers by changing the type of the I4 variable to REAL. This is unportable since floating point numbers do not necessarily have a size of four bytes on all platforms.

Examples (FORTRAN)

This example shows how to open a TIFF file (named example.tif) and read the header. The header of a TIFF file comprises eight bytes:

- Bytes 1, 2: Endian marker (II for little-endian, MM for big-endian);
- Bytes 3, 4: Magic number (42 encoded in a 2-byte integer);
- Bytes 5, 6, 7, 8: The first IFD offset (the byte offset, encoded in a 4-byte integer, of the first set of TIFF data).

```
C
      PROGRAM READ_BIN_EX1
      IMPLICIT NONE
C
      INCLUDE 'readbin.inc'
C
C     Functions
C
      INTEGER OPEN_BIN_FILE
      INTEGER READ_BIN_FILE
      LOGICAL BIGEND_CPU
      INTEGER READ_BIN_SHORT
      INTEGER READ_BIN_LONG
C
C     Variables
C
      BYTE      HEADER(8)        ! Store the 8-byte TIFF header
      INTEGER   MAGIC            ! TIFF magic num. Should be 42
      INTEGER   IFD1_OFFSET      ! The byte offset of the first IFD
C
C     Open TIFF file
C
      IF (OPEN_BIN_FILE(‘example.tif’) .NE. 0) GOTO 901
C
C     Read first 8 bytes of the file into HEADER.
C     N.B. Remember that the start and end bytes are zero-offset
C     meaning that the first byte in the file is byte 0.
C
      IF (READ_BIN_FILE(HEADER, 0, 7) .NE. 0) GOTO 902
C
C     The first two bytes of a TIFF header should be II or MM
C     = 73, 73 or = 77, 77. II means the file is little-endian.
C     MM means the file is big-endian.
C
      IF (HEADER(1) .EQ. 73 .AND. HEADER(2) .EQ. 73) THEN
         BIG_END = .FALSE.
      ELSE IF (HEADER(1) .EQ. 77 .AND. HEADER(2) .EQ. 77) THEN
         BIG_END = .TRUE.
      ELSE
         GOTO 903
      ENDIF
C
C     If the machine has a different endian than the file
C     then SWAP is set to .TRUE.
C
      SWAP = (BIGEND_CPU() .XOR. BIG_END)
      WRITE(*,*) 'BIG_END = ', BIG_END, ' SWAP = ', SWAP
C
C     Check magic number.
C
      MAGIC = READ_BIN_SHORT(HEADER(3))
      WRITE(*,*) 'MAGIC = ', MAGIC
      IF (MAGIC .NE. 42) GOTO 904
C
```

```
C     Read the first IFD offset.
C
      IFD1_OFFSET = READ_BIN_LONG(HEADER(5))
      WRITE(*,*) 'IFD1_OFFSET = ', IFD1_OFFSET
C
      CALL CLOSE_BIN_FILE()
      RETURN
C
C     Error traps
C
 901  WRITE(*,*) '** Error opening TIFF file!'
      RETURN
 902  WRITE(*,*) '** Error reading TIFF header!'
      RETURN
 903  WRITE(*,*) '** Error bad endian marker!'
      RETURN
 904  WRITE(*,*) '** Error bad magic number!'
      RETURN
      END
```

This example shows how to read a ficticious file that has the following format:

- Little-endian data;
- Image size: 625 pixels by 576 pixels;
- 2 bytes per pixel;
- Image data starts at byte 4100.

```
C
      PROGRAM READ_BIN_EX2
      IMPLICIT NONE
C
      INCLUDE 'readbin.inc'
C
C     Functions
C
      INTEGER OPEN_BIN_FILE
      INTEGER READ_BIN_FILE
      LOGICAL BIGEND_CPU
      INTEGER READ_BIN_SHORT
C
C     Variables
C
      INTEGER    NUM_PIX           ! Number of pixels
      PARAMETER (NUM_PIX = 625*576)
      INTEGER    BLOCKSZ           ! Size of the RAWDATA buffer
      PARAMETER (BLOCKSZ = 8192)
      INTEGER    DATA(NUM_PIX)     ! Converted data
      BYTE       RAWDATA(BLOCKSZ)  ! Raw data as read from disc
      INTEGER    I, J              ! Loop counters
      INTEGER    NUM_BLOCKS        ! Number of blocks in file
      INTEGER    BYTESPERPIX       ! Bytes per pixel
      INTEGER    PIXREAD           ! Number of pixels read so far
      INTEGER    OFFSET            ! Byte offset in file to read from
      INTEGER    REMAINDER         ! Number of remaining bytes after
C                                    all blocks have been read
C
C     Open file
C
      IF (OPEN_BIN_FILE(‘example.xxx’) .NE. 0) GOTO 901
C
C     Little-endian file. If the machine has a different endian than
C     the file then set SWAP to .TRUE.
C
      BIG_END = .FALSE.
      SWAP = (BIGEND_CPU() .XOR. BIG_END)
C
C     Start at pixel zero. The data starts at byte 4100.
C     Read data in blocks of BLOCKSZ bytes.
C
      BYTESPERPIX = 2
      PIXREAD     = 0
      OFFSET      = 4100
      NUM_BLOCKS  = NUM_PIX*BYTESPERPIX/BLOCKSZ
      DO I = 1, NUM_BLOCKS
C
         IF (READ_BIN_FILE(RAWDATA, OFFSET,
     $                     (OFFSET+BLOCKSZ-1)) .NE. 0)
     $        GOTO 902
```

```
C
         DO J = 1, BLOCKSZ, BYTESPERPIX
            PIXREAD = PIXREAD + 1
            DATA(PIXREAD) = READ_BIN_SHORT(RAWDATA(J))
         ENDDO
         OFFSET = OFFSET + BLOCKSZ
      ENDDO
C
C     Read the remainder of the data if any
C
      REMAINDER = MOD((NUM_PIX*BYTESPERPIX), BLOCKSZ)
      IF (REMAINDER .NE. 0) THEN
         IF (READ_BIN_FILE(RAWDATA, OFFSET,
     $                     (OFFSET+REMAINDER-1)) .NE. 0)
     $        GOTO 902
C
         DO J = 1, REMAINDER, BYTESPERPIX
            PIXREAD = PIXREAD + 1
            DATA(PIXREAD) = READ_BIN_SHORT(RAWDATA(J))
         ENDDO
      ENDIF
C
C     Close file
C
      CALL CLOSE_BIN_FILE()
C
C     Analysis can now be carried out on the data in DATA.
C
      RETURN
C
C     Error traps
C
 901  WRITE(*,*) '** Error opening file!'
      RETURN
 902  WRITE(*,*) '** Error reading file!'
      RETURN
      END
```

Opening and Reading Binary Files (C)

The C standard library provides function that can be used to open and read binary files: fopen, fread, fseek and fclose. Here is an example the reads and prints out, in hexidecimal, the first eight bytes of a file called ‘example.tif’:

```
#include <stdio.h>
#include <stdlib.h>

#define NUM_BYTES 8

int main(void)
{
    FILE           *file_p;
    int             num_bytes_read;
    int             i;
    unsigned char   raw_data_p[NUM_BYTES];

    /* Open example.tif in read binary mode. */
    if ((file_p = fopen("example.tif", "rb")) == NULL)
    {
        fprintf(stderr, "Error opening file.\n");
        exit(1);
    }

    /* Move file pointer to first byte of file. This is
     * just an example of using fseek. It is not necessary
     * to do this since the file pointer is set to
     * byte zero when the file is opened.
     */
    if (fseek(file_p, 0, SEEK_SET) != 0)
    {
        fprintf(stderr, “Error seeking in file.\n”);
        exit(1);
    }

    /* Read the first NUM_BYTES bytes of the file into raw_data_p */
    num_bytes_read = fread(raw_data_p, sizeof(unsigned char), NUM_BYTES, file_p);

    if (num_bytes_read != NUM_BYTES)
    {
        fprintf(stderr, "Error reading file.\n");
        exit(1);
    }

    for (i = 0; i < NUM_BYTES; i++)
    {
        printf("%x\n", raw_data_p[i]);
    }

    /* Close file. */
    fclose(file_p);
    return 0;
}
```

fopen() takes the file name to open (as a pointer to a NULL-terminated C string—char *) and a string specifying the mode with which to open the file: "rb" mean open in read-only binary mode. fopen() returns a file handle; this is NULL if the open failed.

fclose() takes the file handle of an open file and closes the file.

fseek() moves the file pointer to the specified byte in the file. The next call to fread() will start reading from the new position.

fread() takes a pointer to the buffer that the data are to be read into; the size in bytes of a single item of data; the number of items to read; and the file handle of the opened file to read. fread() returns the number of items read; this may be less the requested number of items if the read failed or the end of the file was reached.

fread() copies the data into the buffer as read from the disc; it does not perform any byte swapping. The function read_bin_file() below wraps fread() and will also swap the bytes in the buffer, if required, before returning. Also included below are two utility functions, swap_2byte() and swap_4byte(), that can be used to swap individual groups of two bytes and four bytes, respectively.

These functions all depend on the readbin.h header file:

```
/**********
 * readbin.h
 */

#ifndef READBIN_H
#define READBIN_H

#include <stdio.h>

typedef enum
{
    MACHINE_BIG_ENDIAN,
    MACHINE_LITTLE_ENDIAN
} e_endian;

typedef enum
{
    DONT_SWAP,
    SWAP
} e_swap;

size_t read_bin_file(void   *buffer_p,
                     size_t  size,
                     size_t  count,
                     FILE   *stream_p,
                     e_swap  swap);
void swap_2bytes(void *raw_data_p);
void swap_4bytes(void *raw_data_p);
e_endian get_endian(void);

#endif /* READBIN_H */
```

```
#include "readbin.h"

/*****************************************************************************
```

```
 *
 * Function: read_bin_file
 *
 * Description: This function wraps the standard library function fread().
 *              It swaps the bytes of the raw data if the parameter swap
 *              is SWAP and size is either 2, 4 or 8. If size is not 2,
 *              4 or 8 then the raw data are passed back unmodified.
 *
 * Input: buffer_p - Pointer the buffer where the data are to be stored.
 *        size     - The size in bytes of items to be read.
 *        count    - The number of items to read.
 *        stream_p - File handle of an open file.
 *        swap     - If SWAP then swap the bytes in the buffer,
 *                   If DONT_SWAP then don't swap bytes.
 *
 * Output: size_t - The actual number of items (not bytes) read. May be less
 *                  than the number requested if an error occurred while
 *                  reading the file or the end of file was reached.
 *
 ***************************************************************************/
size_t read_bin_file(void   *buffer_p,
                     size_t  size,
                     size_t  count,
                     FILE   *stream_p,
                     e_swap  swap)
{
    unsigned char *byte_p;
    unsigned char  tmp;
    size_t         num_items_read;
    size_t         i;

    num_items_read = 0;

    if (stream_p != NULL && buffer_p != NULL)
    {
        num_items_read = fread(buffer_p, size, count, stream_p);

        /* Swap bytes if asked to and if size is divisible by two. */
        if (swap == SWAP && (size % 2 == 0))
        {
            byte_p = (unsigned char *) buffer_p;
            switch (size)
            {
            case 2:
                for (i = 0; i < num_items_read; i++)
                {
                    tmp           = *byte_p;
                    *byte_p       = *(byte_p + 1);
                    *(byte_p + 1) = tmp;
                    byte_p += size;
                }
                break;

            case 4:
                for (i = 0; i < num_items_read; i++)
                {
                    tmp           = *byte_p;
                    *byte_p       = *(byte_p + 3);
                    *(byte_p + 3) = tmp;

                    tmp           = *(byte_p + 1);
                    *(byte_p + 1) = *(byte_p + 2);
                    *(byte_p + 2) = tmp;
                    byte_p += size;
                }
                break;

            case 8:
                for (i = 0; i < num_items_read; i++)
                {
                    tmp           = *byte_p;
                    *byte_p       = *(byte_p + 7);
                    *(byte_p + 7) = tmp;

                    tmp           = *(byte_p + 1);
                    *(byte_p + 1) = *(byte_p + 6);
                    *(byte_p + 6) = tmp;

                    tmp           = *(byte_p + 2);
                    *(byte_p + 2) = *(byte_p + 5);
                    *(byte_p + 5) = tmp;
                    tmp            = *(byte_p + 3);
                    *(byte_p + 3) = *(byte_p + 4);
                    *(byte_p + 4) = tmp;
                    byte_p += size;
                }
                break;

            default:
                fprintf(stderr, "read_bin_file: Can't swap data. ");
                fprintf(stderr, "Unsupported data size: %d\n", size);
                break;
            }
        }
    }

    return num_items_read;
}


/***************************************************************************
 *
 * Function: swap_2bytes
 *
 * Description: Swaps two bytes pointed to by raw_data_p.
 *
 * Input: raw_data_p   - Pointer to the bytes to swap.
 *
 * Output: None.
 *
 ***************************************************************************/
void swap_2bytes(void *raw_data_p)
{
    unsigned char *data_p = (unsigned char *) raw_data_p;
    unsigned char  tmp;

    tmp           = *data_p;
    *data_p       = *(data_p + 1);
    *(data_p + 1) = tmp;
}


/***************************************************************************
 *
 * Function: swap_4bytes
 *
 * Description: Swaps four bytes pointed to by raw_data_p.
 *
 * Input: raw_data_p   - Pointer to the bytes to swap.
 *
 * Output: None.
 *
 ***************************************************************************/
void swap_4bytes(void *raw_data_p)
{
    unsigned char *data_p = (unsigned char *) raw_data_p;
    unsigned char  tmp;

    /* Swap first and fourth bytes */
    tmp           = *data_p;
    *data_p       = *(data_p + 3);
    *(data_p + 3) = tmp;

    /* Swap second and third bytes */
    tmp           = *(data_p + 1);
    *(data_p + 1) = *(data_p + 2);
    *(data_p + 2) = tmp;
}
```

Examples (C)

This example shows how to open a TIFF file (named example.tif) and read the header. The header of a TIFF file comprises eight bytes:

- Bytes 1, 2: Endian marker (II for little-endian, MM for big-endian);
- Bytes 3, 4: Magic number (42 encoded in a 2-byte integer);

- Bytes 5, 6, 7, 8: The first IFD offset (the byte offset, encoded in a 4-byte integer, of the first set of TIFF data).

```
#include <stdio.h>
#include <stdlib.h>
#include "readbin.h"

#define TIFF_BIG_ENDIAN 0x4D4D
#define TIFF_LITTLE_ENDIAN 0x4949
#define TIFF_VERSION 42

typedef struct
{
  unsigned short byte_order;  /* Little-endian or big-endian. */
  unsigned short version;     /* Tiff version number. */
  unsigned int   first_ifd_offset;
} t_tiff_header;

int main()
{
    FILE          *file_p;
    t_tiff_header  tiff_header;
    e_swap         swap_bytes;

    /* Open example.tif */
    if ((file_p = fopen("example.tif", "rb")) == NULL)
    {
        fprintf(stderr, "Error opening file.\n");
        exit(1);
    }

    /* Read byte order marker. */
    if (read_bin_file(&tiff_header.byte_order, sizeof(unsigned short),
                      1, file_p, DONT_SWAP) != 1)
    {
        fprintf(stderr, "Error reading byte order.\n");
        exit(1);
    }

    /* Check byte order marker is valid */
    if (tiff_header.byte_order != TIFF_BIG_ENDIAN &&
        tiff_header.byte_order != TIFF_LITTLE_ENDIAN)
    {
        fprintf(stderr, "Error reading TIFF header: Illegal byte order %hX\n",
                tiff_header.byte_order);
        exit(1);
    }

    /* Work out if we need to swap multibyte sequences */
    if ((tiff_header.byte_order == TIFF_BIG_ENDIAN &&
        get_endian() == MACHINE_LITTLE_ENDIAN) ||
        (tiff_header.byte_order == TIFF_LITTLE_ENDIAN &&
        get_endian() == MACHINE_BIG_ENDIAN))
    {
        swap_bytes = SWAP;
    }
    else
    {
        swap_bytes = DONT_SWAP;
    }

    /* Read TIFF version. */
    if (read_bin_file(&tiff_header.version, sizeof(unsigned short),
                      1, file_p, swap_bytes) != 1)
    {
        fprintf(stderr, "Error reading version.\n");
        exit(1);
    }

    /* Read TIFF version. */
    if (read_bin_file(&tiff_header.first_ifd_offset, sizeof(unsigned int),
                      1, file_p, swap_bytes) != 1)
    {
        fprintf(stderr, "Error reading version.\n");
        exit(1);
    }

     fclose(file_p);
    return 0;
}
```

This example shows how to read a ficticious file that has the following format:

- Little-endian data;
- Image size: 625 pixels by 576 pixels;
- 2 bytes per pixel;
- Image data starts at byte 4100.

```
#include <stdio.h>
#include <stdlib.h>
#include "readbin.h"

#define WIDTH 625
#define HEIGHT 576

int main()
{
    FILE           *file_p;
    unsigned short *data_p;
    e_swap          swap_bytes;

    /* Allocate memory for data. */
    data_p = (unsigned short *) malloc(sizeof(unsigned short)*WIDTH*HEIGHT);
    if (data_p == NULL)
    {
        fprintf(stderr, "Couldn't allocate memory for data.\n");
        exit(1);
    }

    /* Open example.xxx */
    if ((file_p = fopen("example.xxx", "rb")) == NULL)
    {
        fprintf(stderr, "Error opening file.\n");
        exit(1);
    }

    /* Little-endian file: Work out if we need to swap multibyte sequences */
    if (get_endian() == MACHINE_BIG_ENDIAN)
    {
        swap_bytes = SWAP;
    }
    else
    {
        swap_bytes = DONT_SWAP;
    }

    /* Data start at byte 4100. */
    if (fseek(file_p, 4100, SEEK_SET) != 0)
    {
        fprintf(stderr, "Error seeking in file.\n");
        exit(1);
    }

    /* Read data. */
    if (read_bin_file(data_p, sizeof(unsigned short), WIDTH*HEIGHT,
                      file_p, swap_bytes) != WIDTH*HEIGHT)
    {
        fprintf(stderr, "Error reading data.\n");
        exit(1);
    }

    fclose(file_p);
    return 0;
}
```

Conclusion

Using the tools in this article it should be possible to read any binary file format. Once the data are in memory, custom analysis can proceed. The reverse step, writing binary files, may also be of interest to people interested in converting one file format to another. Writing binary files is not much more difficult than reading binary files, however, a description of a binary file writer is outside the scope of this article.

The source code in this article can be found under the 'readbin' directory at:
http://www.ccp14.ac.uk/ccp/web-mirrors/scott-belmonte-software/

cctbx news

R.W. Grosse-Kunstleve, N.K. Sauter & P.D. Adams
Lawrence Berkeley National Laboratory, One Cyclotron Road, BLDG 4R0230, Berkeley, CA 94720-8235, USA. E-mail: RWGrosse-Kunstleve@lbl.gov ; WWW: http://cci.lbl.gov/ and http://cctbx.sourceforge.net/

1: Introduction

The Computational Crystallography Toolbox (cctbx, http://cctbx.sourceforge.net/) is an open-source library of reusable crystallographic algorithms. In this article we give an overview of recent developments. All example scripts shown below were tested with cctbx build 2004_01_16_1718.

2: Reduced cell computations

In the International Tables for Crystallography Volume A an entire chapter (No. 9) is devoted to the discussion of *Crystal Lattices*. At the center of the chapter is a treatment of *reduced bases*. By definition the basis vectors of a reduced basis are the three shortest, non-coplanar lattice vectors. Finding such a basis given a different choice of basis vectors is the subject of cell reduction algorithms. For example, the following `primitive_setting` is the result of transforming a C-centred monoclinic cell:

```
from cctbx import crystal

print "Monoclinic C-centred setting:"
monoclinic_c = crystal.symmetry(
  unit_cell=(25.0822, 5.04549, 29.4356, 90, 103.108, 90),
  space_group_symbol="C12/m1")
monoclinic_c.show_summary()
print "Number of lattice translations:", \
  monoclinic_c.space_group().n_ltr()
print

print "Primitive setting:"
primitive_setting = monoclinic_c.change_basis("-x-y,-x+y,-z")
primitive_setting.show_summary()
print "Number of lattice translations:", \
  primitive_setting.space_group().n_ltr()
```

Output:

```
Monoclinic C-centred setting:
Unit cell: (25.0822, 5.04549, 29.4356, 90, 103.108, 90)
Space group: C 1 2/m 1 (No. 12)
Number of lattice translations: 2

Primitive setting:
Unit cell: (12.7923, 12.7923, 29.4356, 102.846, 102.846, 22.7475)
Space group: Hall: -C 2y (x-y,x+y,z) (No. 12)
Number of lattice translations: 1
```

In this case the intention was to continue a structure refinement in a triclinic space group. However, the transformation `"-x-y,-x+y,-z"` leads to an unfortunate γ angle of about 23° which makes it difficult to visualize the structure. This problem can be avoided by transforming to the corresponding reduced cell:

```
print "Niggli cell:"
niggli_cell = primitive_setting.niggli_cell()
niggli_cell.show_summary()
```

Output:

```
Niggli cell:
Unit cell: (5.04549, 12.7923, 29.3711, 77.7182, 85.0727, 78.6263)
Space group: Hall: -C 2y (-x+y,z,2*x-z) (No. 12)
```

To facilitate this calculation the `uctbx` (unit cell toolbox) module of the cctbx was expanded to include several cell reduction algorithms: a Buerger reduction according to Gruber (1973), a Niggli reduction according to Krivy & Gruber (1976) and a new *minimum reduction* according to Grosse-Kunstleve et al. (2004). The example above shows that the change-of-basis operator which transforms the given basis to the reduced basis is also automatically applied to the space group symmetry (note the change from `C 1 2/m 1` to `Hall: -C 2y (-x+y,z,2*x-z)`). The automatic transformation mechanism extends to entire structures and reflection data:

```
from cctbx import xray
from cctbx.array_family import flex

structure_monoclinic_c = xray.structure(
  crystal_symmetry=monoclinic_c,
  scatterers=flex.xray_scatterer((
    xray.scatterer(label="Si", site=(0.1,0.2,0.3)),
    xray.scatterer(label="O", site=(0.2,0.3,0.4)))))
structure_monoclinic_c.show_summary().show_scatterers()
print

structure_niggli_cell = structure_monoclinic_c.niggli_cell()
structure_niggli_cell.show_summary().show_scatterers()
```

Output:

```
Number of scatterers: 2
At special positions: 0
Unit cell: (25.0822, 5.04549, 29.4356, 90, 103.108, 90)
Space group: C 1 2/m 1 (No. 12)
Label, Scattering, Multiplicity, Coordinates, Occupancy, Uiso
Si   Si     8 ( 0.1000  0.2000  0.3000) 1.00 0.0000
O    O      8 ( 0.2000  0.3000  0.4000) 1.00 0.0000

Number of scatterers: 2
At special positions: 0
Unit cell: (5.04549, 12.7923, 29.3711, 77.7182, 85.0727, 78.6263)
Space group: Hall: -C 2y (-x+y,z,2*x-z) (No. 12)
Label, Scattering, Multiplicity, Coordinates, Occupancy, Uiso
Si   Si     4 (-0.3000 -0.1000  0.3000) 1.00 0.0000
O    O      4 (-0.5000  0.0000  0.4000) 1.00 0.0000
```

Now for reflection data:

```
f_calc_monoclinic_c = abs(structure_monoclinic_c.structure_factors(
  d_min=1, # high-resolution limit
  anomalous_flag=False,
  algorithm="direct").f_calc())
f_calc_monoclinic_c.show_summary()
print

f_calc_niggli_cell = f_calc_monoclinic_c.niggli_cell()
f_calc_niggli_cell.show_summary()
```

Output:

```
Type of data: double, size=2168
Type of sigmas: None
Number of Miller indices: 2168
Anomalous flag: 0
Unit cell: (25.0822, 5.04549, 29.4356, 90, 103.108, 90)
Space group: C 1 2/m 1 (No. 12)

Type of data: double, size=2168
Type of sigmas: None
Number of Miller indices: 2168
Anomalous flag: 0
Unit cell: (5.04549, 12.7923, 29.3711, 77.7182, 85.0727, 78.6263)
Space group: Hall: -C 2y (-x+y,z,2*x-z) (No. 12)
```

This example also shows that the intermediate transformation (`"-x-y,-x+y,-z"` above) is not needed. The change-of-basis operator which transforms a given setting to a setting in the Niggli cell is determined automatically by the unit cell toolbox and not even presented to the user in the example above. However, if so desired the operator is of course available:

```
print "Change-of-basis original cell -> Niggli cell:"
change_of_basis_op = structure_monoclinic_c \
  .change_of_basis_op_to_niggli_cell()
print "  operator:", change_of_basis_op.c()
print "   inverse:", change_of_basis_op.c_inv()
```

Output:

```
Change-of-basis original cell -> Niggli cell:
  operator: -x-y,2*x-z,z
   inverse: 1/2*y+1/2*z,-x-1/2*y-1/2*z,z
```

Our recent paper on reduced cell algorithms (Grosse-Kunstleve et al., 2004) contains pointers to the relevant source code files in the cctbx module.

3: Determination of lattice symmetry

To support a novel auto-indexing procedure that was developed in our group (Sauter et al., 2004), we have added an algorithm for the determination of the lattice symmetry to the `sgtbx` (space group toolbox) module of the cctbx (determining the lattice symmetry is equivalent to determining the Bravais type). The algorithm can be executed via the web interface at http://cci.lbl.gov/cctbx/ or through the command line interface provided in the `iotbx` (input output toolbox) module of the cctbx. For example:

```
% iotbx.lattice_symmetry --unit_cell "12,13,14,92,88,100"

Input
=====

Unit cell: (12, 13, 14, 92, 88, 100)
Space group: P 1 (No. 1)

Angular tolerance: 3.000 degrees

Similar symmetries
==================

Symmetry in minimum-lengths cell: P 1 1 2/m (No. 10)
      Input minimum-lengths cell: (12, 13, 14, 88, 88, 80)
           Symmetry-adapted cell: (12, 13, 14, 90, 90, 80)
            Conventional setting: P 1 2/m 1 (No. 10)
                       Unit cell: (12, 14, 13, 90, 100, 90)
                 Change of basis: -x,-z,-y
                         Inverse: -x,-z,-y
      Maximal angular difference: 2.611 degrees

Symmetry in minimum-lengths cell: P -1 (No. 2)
      Input minimum-lengths cell: (12, 13, 14, 88, 88, 80)
           Symmetry-adapted cell: (12, 13, 14, 88, 88, 80)
            Conventional setting: P -1 (No. 2)
                       Unit cell: (12, 13, 14, 88, 88, 80)
                 Change of basis: -x,y,-z
                         Inverse: -x,y,-z
      Maximal angular difference: 0.000 degrees
```

The first step of the algorithm is to determine a reduced basis as presented in the previous section. The second step is to determine all two-fold axes of the crystal lattice according to Le Page (1982). This is equivalent to searching for 90° angles between lattice vectors. To account for experimental uncertainties and rounding errors it is necessary to consider an `Angular tolerance` (δ_{max} in Le Page, 1982). In the example above deviations up to 3° are permitted. The list of two-fold axes found in the search is sorted by the angular deviation, with the smallest deviation first.

The highest-symmetry space group compatible with the given unit cell is determined by successive group multiplication starting with the first two-fold in the list. The other two-folds are added to the group one-by-one until the list is exhausted or the group multiplication leads to an infinite group (e.g. performing group multiplication starting with a four-fold axis and a six-fold axis leads to an infinite group). This procedure leads to the highest-symmetry space group because any crystallographic space group can be generated using just the two-fold axes and, for centric space groups, a centre of inversion. For example:

```
from cctbx import sgtbx
group = sgtbx.space_group() # start with P1
for two_fold in ["-x,-y,z", "-y,-x,-z", "-z,-y,-x"]:
  group.expand_smx(two_fold)
print sgtbx.space_group_info(group=group)

group.expand_smx("-x,-y,-z") # centre of inversion
print sgtbx.space_group_info(group=group)
```

Output:

```
P 4 3 2
P m -3 m
```

The source code that implements the search for two-fold axes and the group multiplication can be found in the file `cctbx/sgtbx/lattice_symmetry.cpp`.

It may happen that the highest-symmetry space group as obtained by the group multiplication has a fairly large maximum angular deviation (e.g. `2.611 degrees` in above). It is therefore of interest to compute the maximum angular deviation for each subgroup of the highest symmetry. The search for subgroups is quite simple because it is well known that any space group can be generated by performing a group multiplication starting with just two symmetry operations and, for centric space group, the center of inversion. Since any crystal lattice has a center of inversion it can be factored out of the determination of the subgroups. We can work with the acentric subgroup of the highest symmetry and add the center of inversion at the end of the procedure. A two-deep loop, each over all symmetry operations of the highest-symmetry space group, produces all possible combinations of generators for the subgroups. For example:

```
highest_symmetry = sgtbx.space_group_info("P 4 3 2").group()
for i_smx in xrange(highest_symmetry.order_p()):
  for j_smx in xrange(i_smx, highest_symmetry.order_p()):
    subgroup = sgtbx.space_group() # start with P1
    subgroup.expand_smx(highest_symmetry(i_smx))
    subgroup.expand_smx(highest_symmetry(j_smx))
```

In the worst case (cubic symmetry) this involves (24+1)*24/2 = 300 iterations since we are always working with primitive settings and acentric groups; i.e. it is always very fast.

In the complete implementation (file `cctbx/cctbx/sgtbx/subgroups.py`) duplicate subgroups are removed. The unique subgroups are sorted and the maximum angular deviation computed for each. These values can be used as a guide for deciding which symmetry to work with in higher-level applications.

The next step in the determination of all possible lattice symmetries is to make the input unit cell parameters exactly fit the given subgroup symmetry. Symmetry-adapted parameters are obtained by converting the unit cell parameters to a metrical matrix `g` (also known as metric tensor) which must be invariant under the following tensor transformation for all rotation matrices `r` in the subgroup:

```
g = r.transpose() * g * r
```

This follows directly from the transformation law for tensors (e.g. Giacovazzo, 1992, p. 130). For example:

```
from cctbx import sgtbx
from cctbx import uctbx
from cctbx import matrix

space_group = sgtbx.space_group_info("P 3").group()
unit_cell = uctbx.unit_cell("10,10,12,90,90,120")
g = matrix.sym(unit_cell.metrical_matrix())
for s in space_group:
  r = matrix.sqr(s.r().num()) # rotation part of symmetry operation
  print (r.transpose() * g * r).mathematica_form(
    one_row_per_line=True,
    format="%.2f")
```

Output:

```
{{100.00, -50.00, 0.00},
  {-50.00, 100.00, 0.00},
  {0.00, 0.00, 144.00},
}
{{100.00, -50.00, 0.00},
  {-50.00, 100.00, -0.00},
  {0.00, -0.00, 144.00},
}
{{100.00, -50.00, -0.00},
  {-50.00, 100.00, 0.00},
  {-0.00, 0.00, 144.00},
}
```

To see what happens if the unit cell parameters are not compatible with the symmetry we change the b-axis from `10` to `11`:

```
unit_cell = uctbx.unit_cell("10,11,12,90,90,120")
```

New output:

```
{{100.00, -55.00, 0.00},
  {-55.00, 121.00, 0.00},
  {0.00, 0.00, 144.00},
}
{{121.00, -66.00, 0.00},
  {-66.00, 111.00, -0.00},
  {0.00, -0.00, 144.00},
}
{{111.00, -45.00, -0.00},
  {-45.00, 100.00, 0.00},
  {-0.00, 0.00, 144.00},
}
```

An obvious way to make the parameters compatible is to average the transformed metrical matrices and to compute new unit cell parameters from the average (see also Grosse-Kunstleve et al., 2002, section 3.3). The cctbx provides an easy to use interface to this functionality:

```
print space_group.average_unit_cell(unit_cell)
```

Output:

```
(10.5198, 10.5198, 12, 90, 90, 120)
```

As the final step each subgroup along with its symmetry-adapted unit cell is transformed from the primitive setting to a standard setting. This is achieved using the algorithm of Grosse-Kunstleve (1999). The script that puts all steps of the determination of the lattice symmetry together is in the file `iotbx/iotbx/command_line/lattice_symmetry.py`.

4: N-Gaussian approximations to scattering factors

To compute X-ray structure factors a program must have access to the scattering factors of all the elements and ions involved. These data are tabulated in two forms in the International Tables for Crystallography, Volume C, section 6.1.1:

- Primary data: tables of sin(theta)/lambda and the corresponding scattering factor.
- Gaussian approximations to the primary data with 4 Gaussian terms `a * exp(-b * (sin(theta)/lambda)**2)` plus a constant term.

The Gaussian approximations are used by most crystallographic applications (e.g. SHELX, CCP4 and CNS). In general the approximations are valid up to sin(theta)/lambda = 2Å^{-1} (d_{min} = 1/4 Å). Waasmeier & Kirfel (1995) introduced approximations with 5 Gaussian terms plus a constant term that are valid up to sin(theta)/lambda = 6 Å^{-1} (d_{min} = 1/12 Å). Both libraries of approximations have been part of the cctbx for a long time (`cctbx.eltbx.xray_scattering.it1992` and `cctbx.eltbx.xray_scattering.wk1995`).

When computing structure factors for macromolecular structures the high-resolution limit is usually significantly lower than the limit of the Gaussian approximations. At the same time the number of terms plus one for the constant term in the Gaussian approximations enters, to a first approximation, as a linear factor into the CPU time required for a structure factor calculation using the FFT-based algorithm of Ten Eyck (1977). This has prompted Agarwal (1978) to suggest 2-term Gaussian approximations for the most prevalent elements in proteins: H, C, N, O, S. These approximations are valid to d_{min} = 1.5 Å with a relative error of about 1% at the highest resolution. Using these coefficients instead of the 4-plus-1-term approximations of the International Tables reduces the CPU time for sampling the electron density according to Ten Eyck (1977) by about 60%. Since the sampling is one of the rate-limiting steps in macromolecular structure refinement the gain is in practice very substantial.

To facilitate a dynamic adjustment of the number of Gaussian terms we have computed a comprehensive library of N-Gaussian approximations to the primary data for all elements and ions listed in section 6.1.1 of the International Tables. For example:

```
scattering_type: N
stol: 6.00 # d_min: 0.08, max_error: 0.0017
a: 2.7754532 1.3759575 1.0628956 1.038057 0.62582183 0.12084177
b: 15.064476 7.1774688 0.52744677 37.962277 0.18761875 0.047184388
c: 0
stol: 5.00 # d_min: 0.10, max_error: 0.0038
a: 3.2903774 1.8375163 1.0084335 0.62711549 0.23302095
b: 10.300994 30.499187 0.28689128 0.76591255 0.068219922
c: 0
stol: 3.00 # d_min: 0.17, max_error: 0.0081
a: 3.1621224 1.9855589 1.0798456 0.76723966
b: 9.9408274 29.234168 0.57566882 0.15176128
c: 0
stol: 1.70 # d_min: 0.29, max_error: 0.0097
a: 2.9995494 2.2558389 1.7278842
b: 23.27268 7.4543309 0.31622488
c: 0
stol: 0.50 # d_min: 1.00, max_error: 0.0071
a: 4.0103186 2.9643631
b: 19.971888 1.7558905
c: 0
stol: 0.17 # d_min: 2.94, max_error: 0.0088
a: 6.9671502
b: 11.43723
c: 0
```

In this example the 1-term approximation is valid up to d_{min} = 2.94 Å, the 2-term approximation up to d_{min} = 1.00 Å, etc. `max_error` is the maximum relative error over the entire resolution range from sin(theta)/lambda = 0 Å^{-1} up to the `stol` shown above. The limits for the 1-term approximations for the most prevalent elements in protein structures are:

```
H: stol: 0.17 # d_min: 2.94, max_error: 0.0096
C: stol: 0.15 # d_min: 3.33, max_error: 0.0098
N: stol: 0.17 # d_min: 2.94, max_error: 0.0088
O: stol: 0.19 # d_min: 2.63, max_error: 0.0082
S: stol: 0.15 # d_min: 3.33, max_error: 0.0093
```

This table shows that 1-term approximations are fully sufficient at a resolution of 3.5 Å. The limits for the 2-term approximations are:

```
H: stol: 0.42 # d_min: 1.19, max_error: 0.0064
C: stol: 0.50 # d_min: 1.00, max_error: 0.0100
N: stol: 0.50 # d_min: 1.00, max_error: 0.0071
O: stol: 0.55 # d_min: 0.91, max_error: 0.0072
S: stol: 0.55 # d_min: 0.91, max_error: 0.0088
```

This table shows that the vast majority of protein structures can be refined using just 2-Gaussian approximations.

The library of N-Gaussian approximations is automatically used if structure factors are computed via the high-level interface (e.g. `f_calc_monoclinic_c` in the first section above). The given d_{min} is used to dynamically select the approximation with the least number of terms but a maximum relative error of less than 1%. To give an example, the savings in CPU time for sampling the electron density of the structure with the PDB access code 1HGE are:

```
Number of scatterers: 15549
Number of reflections:
  d_min=4: 39137
  d_min=3: 92401
  d_min=2: 310603
  d_min=1: 2474361

dynamic/4-plus-1 using the exp function:
  d_min=4: 0.42 s / 1.28 s = 0.33
  d_min=3: 1.09 s / 2.70 s = 0.40
  d_min=2: 2.61 s / 5.67 s = 0.46
  d_min=1: 20.75 s / 40.22 s = 0.52

dynamic/4-plus-1 using an exp table:
  d_min=4: 0.38 s / 0.91 s = 0.41
  d_min=3: 0.84 s / 1.86 s = 0.45
  d_min=2: 1.95 s / 3.92 s = 0.50
  d_min=1: 14.08 s / 27.22 s = 0.52
```

These times were collected on a 2.8 GHz Xeon computer running Windows 2000. Strictly speaking the absolute times are meaningless without the exact definition of all parameters used in the sampling of the electron density (which is beyond the scope of this article), but the parameters used are typical and the ratios shown above are roughly invariant under a change of parameters. As a rule of thumb, the sampling procedure is about twice as fast if the dynamically selected N-Gaussian approximations are used instead of the 4-plus-1 approximations from the International Tables. The results shown above can be reproduced by running the `electron_density_sampling.py` script in the directory `cctbx/cctbx/development`.

The library of N-Gaussian approximations can also be accessed at a lower level. For example:

```
from cctbx.eltbx import xray_scattering
for n_terms in [1,2]:
  table_entry = xray_scattering.n_gaussian_table_entry("C", n_terms)
  print "d_min:", table_entry.d_min()
  print "max_relative_error:", table_entry.max_relative_error()
  n_gaussian = table_entry.gaussian()
  n_gaussian.show()
  print
```

Output:

```
d_min: 3.33333333333
max_relative_error: 0.00975980236455
a: 5.9679281
b: 14.895768
c: 0

d_min: 1.0
max_relative_error: 0.00995574533403
a: 3.5435555 2.4257967
b: 25.623984 1.5036446
c: 0
```

The raw table can be found in the file `cctbx/eltbx/xray_scattering/n_gaussian_raw.cpp`.

5: Fast structure-factor gradients

For the refinement of macromolecular structures it is essential that the structure-factor and structure-factor-gradient calculations are carried out using a Fast Fourier Transform (FFT) based method. Bricogne (2001) uses the wording "spectacular increases in speed" (p. 91) in connection with such methods. The first to introduce FFT gradient calculations into crystallography was Agarwal (1978). Agarwal's original method requires the computation of a FFT for each type of refinable parameter, but "Lifchitz's reformulation" (Bricogne, 2001) removes this requirement and the Agarwal-Lifchitz procedure is routinely used in programs like TNT (Tronrud et al., 1987), CNS (Brunger, 1989) and REFMAC (Murshudov et al., 1997). The same procedure is now also available in the `cctbx.xray.structure_factors` package. Gradients w.r.t the following parameters are supported:

- coordinates
- isotropic displacement parameters ("B-factors")
- anisotropic displacement parameters
- occupancy factors
- dispersive coefficients ("f-prime")
- anomalous coefficients ("f-double-prime")

The procedure is optimized for structures both with and without anomalous scatterers. Gradients w.r.t any combination of parameters may be computed (e.g. only coordinates, or coordinates and displacement parameters simultaneously, etc.). Isotropic and anisotropic displacement parameters may be arbitrarily mixed. The memory required to store the gradients is allocated dynamically if needed. Of course, the procedure is fully scriptable from Python to maximize reusability and flexibility.

The `cctbx.xray.minimization` module is useful as a starting point to explore how the gradient calculations are used. The core calculations are implemented in C++ and can be found in the file `cctbx/include/cctbx/xray/fast_gradients.h`. Exploiting the abstraction facilities provided by C++, the code for the computation of the gradients makes heavy use of the code for the FFT structure factor calculations which is located in the same directory. As a consequence the C++ source code specific to the gradient calculations is only 620 lines long.

The cctbx also includes the much simpler direct-summation procedure for computing structure-factor gradients. Both the FFT-based procedure and the direct-summation procedure are accessible through a uniform Python interface. The desired method is selected via `algorithm="direct"` or `algorithm="fft"` as shown for the calculation of `f_calc_monoclinic_c` in the example in the first section. If `algorithm` is not specified, a heuristic procedure determines automatically which method to use.

6: Universal reflection file reader

To support the substructure determination procedure in Phenix (Grosse-Kunstleve & Adams, 2003) we have implemented a reflection file reader that automatically detects and processes these formats:

```
- merged scalepack files
- unmerged scalepack files
- CCP4 MTZ files with merged data
- CCP4 MTZ files with unmerged data (but merged files are preferred)
- d*trek .ref files
- XDS_ASCII files with merged data
- CNS reflection files
- SHELX reflection files
```

Using this reader is extremely simple:

```
from iotbx import reflection_file_reader
reflection_file = reflection_file_reader.any_reflection_file(
  file_name="gere_MAD.mtz")
miller_arrays = reflection_file.as_miller_arrays()
for miller_array in miller_arrays:
  miller_array.show_summary()
```

A fragment from the output:

```
Miller array info: gere_MAD.mtz:F(+)SEpeak,SIGF(+)SEpeak,F(-)SEpeak,SIGF(-
)SEpeak
Observation type: xray.amplitude
Type of data: double, size=23010
Type of sigmas: double, size=23010
Number of Miller indices: 23010
Anomalous flag: 1
Unit cell: (108.742, 61.679, 71.652, 90, 97.151, 90)
Space group: C 1 2 1 (No. 5)
```

Note that in this case the reader automatically combines four data columns of the input MTZ file into one object. The low-level processing of the reflection data is handled automatically as much as possible using all available information. However, sometimes the reader needs a little more help. For example when reading CNS reflection files the unit cell and space group are not available. Here is how the information can be supplied externally:

```
from iotbx import reflection_file_reader
from cctbx import crystal
reflection_file = reflection_file_reader.any_reflection_file(
  file_name="scale.hkl")
crystal_symmetry = crystal.symmetry(
  unit_cell=(108.742, 61.679, 71.652, 90, 97.151, 90),
  space_group_symbol="C2")
miller_arrays = reflection_file.as_miller_arrays(
  crystal_symmetry=crystal_symmetry)
for miller_array in miller_arrays:
  miller_array.show_summary()
```

Sometimes certain space groups are used as placeholders during data processing until the true space group is known (e.g. P222 instead of P212121). The method above can also be used to replace the information found in the reflection file with the correct symmetry information.

To minimize the need for manual entering of symmetry information, the iotbx provides a facility for extracting just the unit cell parameters and the space group from all reflection file formats shown above (all that actually contain symmetry information) and some other file formats such is CNS input files, SHELX .ins files and SOLVE input files. As before, the format is detected and processed automatically:

```
from iotbx import crystal_symmetry_from_any
crystal_symmetry = crystal_symmetry_from_any.extract_from(
  file_name="shelx.ins")
crystal_symmetry.show_summary()
```

The `crystal_symmetry` object obtained in the example can be used as an argument to the `as_miller_arrays()` method in the previous example.

7: Notes on supported platforms

In regular intervals the cctbx is automatically built on a large number of platforms and easy-to-install binary distributions are posted at http://cci.lbl.gov/cctbx_build/ . Currently, 15 different binary bundles are available (various versions of Windows, Linux, Mac OS X, IRIX, Tru64 Unix combined with different versions of Python). Not all of them are strictly needed. For example the binary bundles for Windows 2000 will also work under Windows XP. However, our approach ensures that the build procedure itself works in all these different environments.

In the last newsletter we announced limited support for Mac OS X. Fortunately the situation has improved significantly since then. The new compilers provided by Apple can now be used under both OS 10.2 and OS 10.3 and the C++ optimizers are fully functional. Python 2.3, which is required for running the cctbx under OS 10, was officially released in July 2003 and is included by default in all OS 10.3 installations.

In addition to the platforms posted at our web site, we have successfully tested the cctbx in the following environments:

- RedHat 8, Intel C++ 7.1.006, native Python (2.2.1)
- RedHat 8, Intel C++ 8.0.058, native Python (2.2.1)
- SunOS 5.9, GCC 3.3.1, Python 2.3 installed automatically from source code bundle
- SuSE SLES-8.1 (AMD64 Opteron), native gcc (3.2.2), native Python (2.2.1)

Currently we are not aware of any major platform where the cctbx could not be used.

8: Acknowledgments

We would like to thank Michael O'Keefe for giving us access to an electronic version of the primary scattering factor data. Wolfgang Kabsch kindly answered our questions regarding the XDS_ASCII reflection file format. We are grateful for the permission to use the CMTZ library provided by CCP4. Our work was funded in part by the US Department of Energy under Contract No. DE-AC03-76SF00098. We gratefully acknowledge the financial support of NIH/NIGMS.

9: References

Agarwal, R.C. (1978). Acta Cryst. A34, 791-809.
Bricogne, G. (2001). In: International Tables for Crystallography, Volume B.
Brunger, A.T. (1989). Acta Cryst. A45, 42-50.
Giacovazzo, C. (1992). Editor. Fundamentals of Crystallography. IUCr/Oxford University Press.
Grosse-Kunstleve, R.W. (1999). Acta Cryst. A55, 383-395.
Grosse-Kunstleve, R.W., Adams, P.D. (2002). J. Appl. Cryst. 35, 477-480.
Grosse-Kunstleve, R.W., Sauter, N.K., Adams, P.D. (2004). Acta Cryst. A60, 1-6.
Gruber, B. (1973). Acta Cryst. A29, 433-440.
Krivy, I. & Gruber, B. (1976). Acta Cryst. A32, 297-298.
Le Page, Y. (1982). J. Appl. Cryst. 15, 255-259.
Murshudov, G.N., Vagin, A.A. & Dodson, E.J. (1997). Acta Cryst. D53, 240-253.
Sauter, N.K., Grosse-Kunstleve, R.W., Adams, P.D. (2004). Submitted to J. Appl. Cryst.
Ten Eyck, L.F. (1977). Acta Cryst. A33, 486-492.
Tronrud, D.E., Ten Eyck, L.F., Matthews, B.W. (1987). Acta Cryst. A43, 489-501.
Waasmeier & Kirfel (1995). Acta Cryst. A51,416-431.

The Once and Everliving FORTRAN : Why Fortran still goes onward and upward while many of its "replacement" languages have already died

Juan Rodríguez-Carvajal,
Laboratoire Léon Brillouin (CEA-CNRS), CEA/Saclay, 91191 Gif sur Yvette Cedex, FRANCE, E-mail: juan@llb.saclay.cea.fr *, WWW:* http://www-llb.cea.fr/fullweb/

Abstract

Most Computer Science gurus have considered Fortran as an “old” and “limited language”. It was predicted to die in many occasions and to be replaced by more “sexy” languages. The fact is that Fortran has evolved taking from other languages all the good concepts and constructs appropriate for scientific computing. The evolution has proceeded by adding new features and maintaining compatibility with older versions in order to preserve the previous investment in software development. This is, in my opinion, the reason of the survival, and even the vitality, of Fortran in present days. I describe in this article the evolution and the present situation of the Fortran language and how the new developments can help crystallographic software.

Introduction

In this section I first give a very short summary of the Fortran language history. To get more information about the Fortran language one can consult the site of reference [1], where many links of interest to Fortran may be found.

Fortran, which is short for “FORmula TRANslation”, was the first successful high-level language (HLL) that allowed to avoid coding in assembler. It was developed over a three-year period (1954-1957) by a team at IBM lead by John Backus. Among the members of the team was the well-known crystallographer David Sayre. The first Fortran compiler (FORTRAN I) and its descendents (up to FORTRAN IV) were extremely successful in the scientific and engineering community.

On May 1962 another milestone was traversed, an ASA committee started developing a standard for the FORTRAN language, a very important step that made it worthwhile for vendors to produce FORTRAN systems for every new computer, and made FORTRAN an even more popular HLL. The new ASA standard was published in 1966, and was known accordingly as FORTRAN 66 (mostly FORTRAN IV), it was the first HLL standard in the world.

By the 1970s it had started to show its age in relation to some of the other languages that had emerged and there was pressure to incorporate changes to bring it more into line with mainstream language development. The next version, standardized in 1978, was called Fortran 77 (notice the lower case spelling that we use hereafter to refer post-77 Fortran), and whilst the new changes were on the welcome side, many felt that they had not gone far enough.

The next Fortran standard (Fortran 90) was published too many years after Fortran 77 was out, allowing other programming languages to evolve and compete with Fortran. For example, the system-programming language C, and its evolved variant C++, started to be also popular in the traditional strong-holds of Fortran. A part of C and C++ other languages emerged and established themselves: Pascal, Ada, Modula-2, Java, etc… all became rivals to Fortran in the scientific and academic communities. In spite of the development of all these new languages, Fortran is still living, whereas some of their competitors died in the sense that they have evolved to new incompatible languages or are seldom used in the scientific community. The general availability of PC computers has had, as a consequence, a huge impact in the nature of the programs written for being run in those machines. The scientific community is no more dominant as the main user of computing resources, so many special languages and developing tools exist in the market. In this situation, it is clear that Fortran is not the language of which one will find manuals in computing drugstores, but it is still the most used language in numerical applications.

The current standard Fortran 95, a minor extension to Fortran 90, includes constructions imported from High Performance Fortran that is an *ad hoc* extension developed for parallel computing. Presently there are two subsets (ELF90 and F) of Fortran 95 that conserve only the newest features of Fortran.

The draft of the new, Fortran 2003, standard has been published in October 2003 and its definitive approval and publication will be during 2004. Fortran 2003 constitutes a major breakthrough in the development of Fortran. It includes the features of the modern Object Oriented Programming (OOP) paradigm conserving the Module Oriented Programming (MOP) introduced with the Fortran 90 standard, together with improvement and additions of features relevant to reliable numerical computing.

In the rest of the paper we give a more detailed description of some elements of the language giving examples of interest for Crystallography.

The Influence of Other Languages and Features of Modern Fortran

After the release of the first commercial Fortran compiler in 1957, other projects for development of computing languages were set up. The most important was the ALGOL (ALGOrithmic Language) project that gave rise to the standard of the 2nd generation of high-level languages. ALGOL, like FORTRAN, was also designed specifically for programming scientific computations. However the international committee that defined the language considered ALGOL as a universal language. Although ALGOL never reached the level of commercial popularity of FORTRAN, it is considered the most important language of its era in terms of its influence on later language development.

Niklaus Wirth, which was involved in the definition of the language by membership of the ALGOL committee, was at the origin of the paradigm of Structured Programming (SP) that dominated the computing science world in the late seventies and the eighties. Wirth is the creator of Pascal, which is the ancestor of the most prolific and interesting set of computing languages ever seen. Wirth is also the main architect of Modula-2 and Oberon that have influenced some aspects of modern Fortran.

In figure 1 one can see an informal list of computing languages revealing some of their filiations and relationships. Arrows indicate the sense of interactions or ancestor/descendent relations.

It is interesting to notice that, within the computer science community, the Pascal family was dominating teaching of programming and the description of algorithms from the middle of the seventies. The ideas of SP influenced Fortran with the introduction, in Fortran 77, of the constructions like IF-THEN-ELSE that helped to avoid “spaghetti code”, typical of early Fortran programming, made of intricate “calculated IFs” and “GOTOs”. The modifications of the Fortran 66 into the Fortran 77 standard were quite conservative, however the scientific community continued to use Fortran in spite of their intrinsic limitations. Of course, the main reason was that the Fortran compilers produced much more efficient executable codes, for numerical problems treated by physicists/chemists and engineers, than those for any other language.

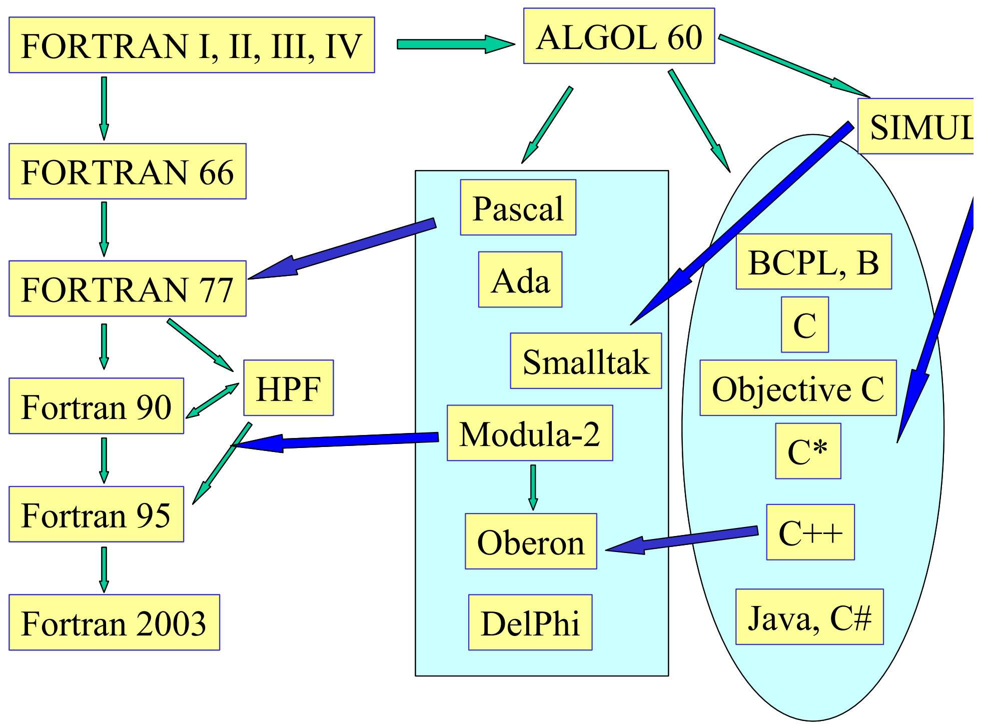


Figure 1: Informal list of programming languages in relation to the Fortran suite of standards. Only the Pascal and C families of ALGOL 60 are considered in some details.

The SP paradigm and their corresponding languages (Pascal family) was a very rich niche of ideas but remained, to a great extend, associated to the academic world. On the contrary, the other major family (C-family) of computing languages originating from the ALGOL project, started as a practical exercise for rewriting the UNIX operating system. The C programming language, dominates from the beginning of the eighties, the world of computer system development. The success of C was due to the fact that it acted as a portable super-assembler language. C was quite close to the machine and the programmers involved in the writing of operating systems and compilers obviously adopted this extremely useful and compact language for those purposes. The C language is at the heart of UNIX-like and Windows operating systems that dominate presently the computing world.

The C language has several drawbacks as a HLL. Apart from a more cryptic syntax than those of the Fortran and Pascal family, the most important drawback concerns a crucial point related to optimization. In C, at difference with Fortran, pointers are always exposed in the language (for instance, to pass the argument by reference, that is transparent in Fortran, one need to use pointer syntax e.g. *arg) and no restriction exists in using pointer arithmetic and the kind of object to be pointed. This gives rise to a great complexity and compilers are unable to produce completely optimized code. In words of Dennis M. Ritchie, the creator of the C language [2]:

On the other hand, C's treatment of arrays in general (not just strings) has unfortunate implications both for optimization and for future extensions. The prevalence of pointers in C programs, whether those declared explicitly or arising from arrays, means that optimizers must be cautious, and must use careful dataflow techniques to achieve good results. Sophisticated compilers can understand what most pointers can possibly change, but some important usages remain difficult to analyze. For example, functions with pointer arguments derived from arrays are hard to compile into efficient code on vector machines, because it is seldom possible to determine that one argument pointer does not overlap data also referred to by another argument, or accessible externally. More fundamentally, the definition of C so specifically describes the semantics of arrays that changes or extensions treating arrays as more primitive objects, and permitting operations on them as wholes, become hard to fit into the existing language. … Thus, C covers the most important uses of strings and arrays arising in practice by a uniform and simple mechanism, but leaves problems for highly efficient implementations and for extensions.

This limitation has been inherited by its most important descendent: C++. The development of the C family of languages has been marked by the incoming of the Object Oriented Programming paradigm that is rooted in the concepts introduced in SIMULA 67 [3]. The C++ language has played the same role with respect to C as Fortran 90 with respect to Fortran 77.

Functionality	**F77**	**C**	**C++**	**F90**
Numerical robustness	**2**	**4**	**3**	**1**
Data parallelism	3	3	3	1
Data abstraction	**4**	**3**	**2**	**1**
Object oriented programming	**4**	**3**	**1**	**2**
Functional programming	4	3	2	1
average	**3.4**	**3.2**	**2.2**	**1.2**

Figure 2: A rough comparison of Fortran 77, C, C++ and Fortran 90 as published, in the middle of 1990s, by the Computational Science Education Project [4]. **The best is noted "1" and the worse "4".**

The first real revolution in the Fortran language was the adoption of the Fortran 90 standard. Even if the standard arrived quite late, nothing was lost. Scientist and engineers working in numerical calculations are quite conservative in using programming, because, contrary to computer scientist, they look at programming *as a tool* for their research: "If something works, why do I have to change?". Despite Fortran 90 compilers exist from many years, most of the computing programs are still written using the Fortran 77 style and rules. The important thing is that Fortran 90 is a superset of Fortran 77 and legacy code can be handled with new compilers. This allows a stepwise way of moving to the new standard without making expensive rewriting and testing of previous well working codes.

Fortran 90/95 is a language that has imported many important concepts from the Pascal family. In particular the concept of Module, at the heart of the Modula-2 language, as a component that acts not only as a static and single instance of a class (in terms of the current OOP jargon) but as a fundamental unit of encapsulation and implicit interface for types, procedures (subroutines and functions), and operators. The introduction of the new features was done preserving the forte of Fortran: fast execution of the generated machine code. In figure 2 we give a comparison of C/C++ and Fortran as published by the Computational Science Education Project [4]

A summary of the new (with respect to Fortran 77) features of the Fortran 90/95 language is give below:

- Free source form:
  - Names can be up to 31 characters in length, blanks are significant.
  - Lines up to 132 characters in length, up to 39 continuation lines, ";" as statement separator for multiple statements per line , "!" as comment symbol
  - Include option for source text from files.
- Modern control structures
  - Fortran 90 has a modern DO statement, with CYCLE and EXIT options and the control part of the DO can be conventional iteration, WHILE or no control clause. There is also a CASE block structure.
  - Intent attributes for arguments of procedures.
- Specification of numeric precision. There is now a clean way to control numeric precision.
- Whole array processing
  - It is now possible to treat arrays as whole objects and simply write A=B*SIN(A), or A=matmul(B,C) where A, B and C are arrays of the adequate shape and size for the demanded operations.
  - FORALL, WHERE constructs

- Dynamic behavior, including allocate, deallocate, pointers, recursion.
- User defined data types and operators.
- Modules, and with them come
    - operator overloading
    - generic procedures

The new features of the language bring it up to date and increase the range of problems that can be solved easily. Fortran 90/95 is not a OO language but we notice in the above list that the language contains the first steps to OOP with encapsulation and a limited support for polymorphism. Fortran 90/95 lacks of direct dynamic polymorphism and inheritance, but it may be introduced by software emulation. See [5] for OOP with Fortran 90.

The full Fortran 90/95 compilers are quite complex due to the need of backward compatibility demanded by the standard. In the site [1] one can find how to get Fortran 95 compilers. For working with legacy code one can just compile it and work with the executable generated code, or (if development is worthwhile) transform it to free format source and progressively replace COMMON blocks by modules, use the IMPLICIT NONE statement, use the INTENT attribute for all procedure arguments, etc. To help do this task, free programs can be downloaded from [6]. The benefit of doing that is to catch many hidden errors at compile time and all the problems related with alignment of COMMON blocks disappear forever.
One can, however, use for new programs just the modern features trying to encapsulate all related procedures, types and operators in modules. For that there exist a very strict subset of Fortran 90/95. This is the F-language, which is freely available for the most important operating systems [7]. As is grandfather, Fortran, the F-language is not designed within the full OOP paradigm but it constitutes a very simple language to start with. The learning curve for F is much less steeper than for any other language of similar powerfulness. The position of F with respect of Fortran is similar of Java with respect to C++ concerning the simplifications adopted in the language [8]. Java is much more simple than C++ but a C++ compiler is unable to compile Java code. On the contrary, whatever Fortran compiler can handle F-programs because F is just a subset of standard conforming Fortran.

The forthcoming Fortran 2003

High Performance, Scientific and Engineering Computing:
Asynchronous I/O, IEEE arithmetic, floating point exception handling, interval arithmetic

Data Abstraction / User Extensibility:
Allocatable components, derived type I/O, parameterised derived types, mixed component accessibility, public entities of private types, ASSOCIATE construct, generalized constructors

Object-oriented Fortran:
Extensible types, inheritance, polymorphism, dynamic type-allocation, type-bound procedures
Procedure pointers

Other Features:
Internationalization, Interoperability with C

Figure 3: List of the most important new features of Fortran 2003

Many of the new features concern object orientation. In figure 3 we show the most important items concerning the new additions to the Fortran standard.

It is worth recognizing that, a part of the OOP features of Fortran 2003, the standardization of how to call C libraries is an important point for extending the low level abilities of the language. The C/Fortran interoperability is important because a significant fraction of computing environments come with a C (API). Examples include X-windows libraries, Motif, OpenGL, TCP/IP socket calls and interfaces to system routines. The Fortran programmer is currently unable to exploit this wealth of software in a portable manner.

A very short and useful document describing the most important new features of Fortran 2003 has been written by John Reid, one of the WG5 conveners. The full document containing the working draft of the Fortran 2003 standard is a PDF document of 585 pages. Both document can be found in the sites of reference [9].

The Fortran language in Crystallography.

The major part of computing programs used in Crystallographic applications is written in Fortran 77. A part from visualization programs, that make use of libraries written mostly in C/C++, and several other programs (FOX, cctbx, Clipper, MAUD…), the dominant language of well known programs (SHELX, PLATON, SIR suite, GSAS, FullProf, etc) is Fortran. It is very common, due to obvious reasons, that Fortran is combined with C/C++ or with scripting/interpreted languages like Python and Tcl/Tk for interactivity and making Graphic User Interfaces (GUI). It is however possible to write full Windows or X-Windows applications using exclusively Fortran. Examples are WinGX, WinPLOTR and GFourier. There are libraries interfacing the low level Windows API making it easy to program a complete GUI in Fortran [10].

The incursion of commercial programs in the arena of crystallographic computing has had as a consequence that the number of crystallographer still developing software has diminished. However, thanks to the activity of some individuals and the CCP14 site [11], the diversity and survival of free development seems, for the moment, to be ensured. There are still many specialized areas, where software development is still worthwhile, in which commercial companies are not too much interested due to the lack of a profitable market.

It is clear that OOP is very useful in Crystallography and some projects, in C++ [12, 13] or modern Fortran [14], are developing adapted tools making use of OO features. We are here concerned with Fortran and I give in figure 3 one example (just a flavor) of what can be done using the F language.

It is interesting to spend few words explaining how it works. The program *uses* six modules from the *CrysFML* library [14]. Not all object and procedures are imported, only those after the clause *only* are used in this small program. The program declares the objects *SpG*, *A* and *Cell* as been of an appropriate *type*, and other auxiliary variables. The *allocatable* array *hkl*, of type *Reflection_Type*, contains as components, the (*hkl*) indices, the value of sinθ/λ, the structure factor, phase, multiplicity, etc.

The program itself is an infinite loop asking for the name of a CIF file. If no name is given the program stops. Once a name is obtained the program calls the procedure *Readn_set_Xtal_Structure*. This procedure read the CIF file and construct the objects *SpG*, *A* and *Cell* needed for further calculations. After checking that there is no error and writing in the structural information in an output file, the program allocates space for the array *hkl* that is partially constructed by calling *Hkl_Uni* (generation and ordering of unique reflections up to a maximum value of sinθ/λ). The procedure *Structure_Factors* is a overloaded subroutine, with several optional arguments, that completes the *hkl* array having the structure factors among its components. Finally the list is written in the output file.

One has to know that using the same name (*Structure_Factors*) one can calculate (omitting the *Mode* argument) the X-ray structure factors and, giving the optional argument *Lambda* (not shown), take into account the anomalous scattering. The same procedure can be invoked by passing an object of type *Molecular_Crystal_Type* instead of the two first arguments (*A* and *SpG*) to calculate the structure factors when the crystal structure is described in terms of a mixture of free atoms and molecular objects.

```
Program Calc_structure_factors

use crystallographic_symmetry, only: space_group_type, Write_SpaceG
use Atom_Module,               only: Atoms_List_Type, Write_Atoms_List
use crystal_types,             only: Crystal_Cell_Type, &
                                          Write_Crystal_Cell
use Reflections_Utilities,     only: Reflection_Type, Hkl_Uni, &
                                     get_maxnumref
use IO_Formats,                only: Readn_set_Xtal_Structure, &
                                     err_mess_form, err_form
use Structure_Factor_Module,   only: Structure_Factors, &
                                     Write_Structure_Factors
type (space_group_type)  :: SpG
type (Atoms_list_Type)   :: A
type (Crystal_Cell_Type) :: Cell
type (Reflection_Type),allocatable, dimension(:) :: hkl
character(len=256)       :: filcod  !Name of the input file
real                     :: stlmax  !Maximum Sin(Theta)/Lambda
integer                  :: MaxNumRef, Num, lun=1

    do
     write(unit=*,fmt="(a)") " => Code of the file xx.cif (give xx): "
     read(unit=*,fmt="(a)") filcod
     if(len_trim(filcod) == 0) exit
     write(unit=*,fmt="(a)") " => Maximum sinTheta/Lambda: "
     read(unit=*,fmt=*) stlmax
     open(unit=lun,file=trim(filcod)//".sfa", &
          status="replace",action="write")

     call Readn_set_Xtal_Structure(trim(filcod)//".cif", &
                                   Cell,SpG,A,Mode="CIF")
     If(err_form) then
       write(unit=*,fmt="(a)") trim(err_mess_form)
       exit
     else
       call Write_Crystal_Cell(Cell,lun)
       call Write_SpaceG(SpG,lun)
       call Write_Atoms_List(A,lun=lun)

       MaxNumRef = get_maxnumref(stlmax,Cell%CellVol,mult=SpG%Multip)
       if(allocated(hkl)) deallocate(hkl)
       allocate (hkl(MaxNumRef))
       call Hkl_Uni(Cell,Spg,.true.,0.0,stlmax,"s",Num,hkl)
       call Structure_Factors(A,SpG,Num,hkl,mode="NUC")
       call Write_Structure_Factors(lun,Num,hkl,mode="NUC")
     end if
     close(unit=lun)
    end do

End Program Calc_structure_factors
```

Figure 3: Text of a simple program calculating nuclear neutron diffraction structure factors. The program uses modules from the *CrysFML* library [14].

In the near future, using the new features available from Fortran 2003, the whole *CrysFML* library can be simplified strongly with straightforward additions and removing pieces of code (mostly by eliminating redundant code due to the lack of dynamic polymorphism in Fortran 95). For example, instead of creating new types, similar to *Space_Group_Type* (*Magnetic_Group_type*, *Group_k_Type*, etc), which contain a component of type *Space_Group_Type*, one has just to add the attribute *extends*(*Space_Group_Type*) to the new types to simplify the access to their components. Moreover the new types import the type-bound procedures of *Space_Group_Type*.

A template of the procedure may be as follows. The declaration of *Space_Group_Type* in the module *Crystallographic_Symmetry* takes the form:

```
Type, public :: Space_Group_Type
   ... ! Declaration of all components
   ...
End Type Space_Group_Type
```

No type-bound procedures is allowed in Fortran 95, in Fortran 2003 we have just to make the change, in a single place, to:

```
Type, public :: Space_Group_Type
   ... ! Declaration of all components
   ...
   contains    !Attach module procedures to the type (Fortran 2003)
     procedure :: Set    => Set_SpaceGroup
     procedure :: writes => Write_SpaceG
     ....
End Type Space_Group_Type
```

The procedures *Set_SpaceGroup* and *Write_SpaceG* are already written in the same module they are renamed and bound to the type in the above construction. So, if the object *SpG* is of this type the procedure *Set* (in fact *Set_SpaceGroup*) can be invoked using the syntax : *call SpG%Set*(…). For details and subtleties see [9].

For instance, the type *Magnetic_Group_type* can be defined as:

```
Type, public, extends(Space_Group_Type):: Magnetic_Group_Type
   Character(len=20)   :: OG_Symbol !Opechowski-Guccione Symbol
   Integer, allocatable, dimension(:) :: time_rev
   contains
     procedure :: Set_TimeRev
     .....
 End Type Magnetic_Group_Type
```

Were we have added two components to the type and a new procedure. One can declare a polymorphic object as:

```
Class(Space_Group_Type):: SpG
```

The object *SpG* can be either of *Space_Group_Type* or of whatever of the types extending *Space_Group_Type*.

The features shortly described in this example, and much more, are available in the new standard [9].

The future of Fortran

"I am not certain what the language for scientific and engineering computation will look like by the 21st century, but I am sure it will be called Fortran." (attributed to C.R.A. Hoare 1982, but probably anonymous)

The Fortran standard (Fortran 95 and soon 2003) provides a modern language with no rival in its application domain: scientific and engineering programming language. Fortran has withstood the test of time because over the years it has modernized in a controlled fashion. The simplicity and clarity of the syntax, the facilities for global array manipulation, the combination of the OOP techniques with MOP (as done for instance in Oberon), the efficiency of the generated executables and the compatibility with legacy code are strong points in favor of a long live to Fortran.

It is worth having a look in the Web pages of companies involved in High Performance Computing (HPC), see, for instance, CRAY Inc and Intel. Let us pick up a small comment from Intel:

"The new products include Intel® C++ and Fortran Compilers for Windows and Linux as well as Intel C++ Compilers for Windows CE .NET.
…The Intel® Visual Fortran Compiler for Windows represents the next generation in compiler technology for high performance computing. …"

The compilers they provide are only for C/C++ and Fortran. The same applies to CRAY Inc. These languages seem to be the only candidates for large scale HPC projects in the long term. C/C++ for its capabilities to access the hardware, and the availability of many visualization libraries, and Fortran for its reliability and efficiency in number crunching applications.

It is clear that the saga of computing languages is not finished but I hope the community of Fortran programmers is strong enough to preserve and maintain (making it evolve) the Fortran language. I'm sure we will be using Fortran for years.

References:

[1] A Web site from which on can obtain a wealth of information about Fortran is http://www.fortran.com. See also http://www.fortran.com/metcalf.htm and
[2] Dennis M. Ritchie, *The development of the C-language* in History of Programming Languages-II ed. Thomas J. Bergin, Jr. and Richard G. Gibson, Jr. ACM Press (New York) and Addison-Wesley (Reading, Mass), 1996; ISBN 0-201-89502-1.
[3] For a very brief introduction to the history of Object Oriented Programming look at the site http://heim.ifi.uio.no/~kristen/FORSKNINGSDOK_MAPPE/F_OO_start.html,
See also: Budd, Timothy. *An Introduction to Object-Oriented Programming* 2nd Edition, Addison-Wesley, ISBN 0-201-82419-1, 1997.
[4] Computational Science Education Project http://csep1.phy.ornl.gov/CSEP/PL/NODE2.html
[5] Object Oriented Programming in Fortran 90: http://www.cs.rpi.edu/~szymansk/oof90.html
See also the book: Object-Oriented Programming via Fortran 90/95 by Ed Akin
[6] Alan Miller site: http://users.bigpond.net.au/amiller/ . See also the program convert from Michael Metcalf: ftp://ftp.numerical.rl.ac.uk/pub/MandR/convert.f90
[7] All free F-compilers can be downloaded from the site: ftp://ftp.swcp.com/~walt/pub/F See also http://www.fortran.com/fortran/Imagine1
[8] Walt Brainerd, David Epstein and Richard Hendrickson, *MacTech Magazine* Vol. 13, No. 9, pages 30-39 (1997). Available at: http://www.fortran.com/F/java.htm
[9] John Reid, WG5 Convener, *The new features of Fortran 2000*, PDF document available directly from the Internet: ftp://ftp.nag.co.uk/sc22wg5/N1451-N1500/N1495.pdf
The full working draft is in: http://www.dkuug.dk/jtc1/sc22/open/n3661.pdf
[10] Collaborative Computational Project Number 14 (CCP14) for *Single Crystal and Powder Diffraction*, http://www.ccp14.ac.uk/
[11] The most popular commercial library interfacing the Windows and X-Windows API is Winteracter. See http://www.winteracter.com/ . For Windows alone RealWin is quite simple an cheap. See http://www.indowsway.com/
[12] Ralf W. Grosse-Kunstleve and Paul D. Adams, *State of the Toolbox: an overview of the Computational Crystallography Toolbox (CCTBX).*
In Compcomm Newsletter No 1. Jan 2003. Available at: http://www.iucr.org/iucr-top/comm/ccom/newsletters/2003jan/
[13] Kevin Cowtan, *The Clipper C++ libraries for X-ray crystallography*
In Compcomm Newsletter No 2. July 2003. Available at: http://www.iucr.org/iucr-top/comm/ccom/newsletters/2003jul/
[14] Juan Rodriguez-Carvajal and Javier González-Platas , *Crystallographic Fortran 90 Modules Library (CrysFML): a simple toolbox for crystallographic computing programs*.
In Compcomm Newsletter No 1. Jan 2003. Available at: http://www.iucr.org/iucr-top/comm/ccom/newsletters/2003jan/

Reprint of the 1947 paper, A Hollerith Punched-card Method for Evaluation of Electron Density in Crystal Structure Analysis, by E.G. Cox, L. Gross and G.A. Jeffrey, Proc. Leeds Phil. Soc., 5, 1-13, (1947), with modern introduction by Professor Stanley Nyburg.

(thanks to Dr Peter Evennett of the Leeds Philosophical and Literary Society, founded in 1819 - http://www.leedsphilandlit.org.uk/, for permission to reprint this paper)

Professor Stanley C. Nyburg,
Chemistry Department, King's College London, Strand, London, England, WC2R 2LS. E-mail: stanley.nyburg@kcl.ac.uk ; WWW: http://www.ch.kcl.ac.uk/kclchem2/academic/asscn1.htm

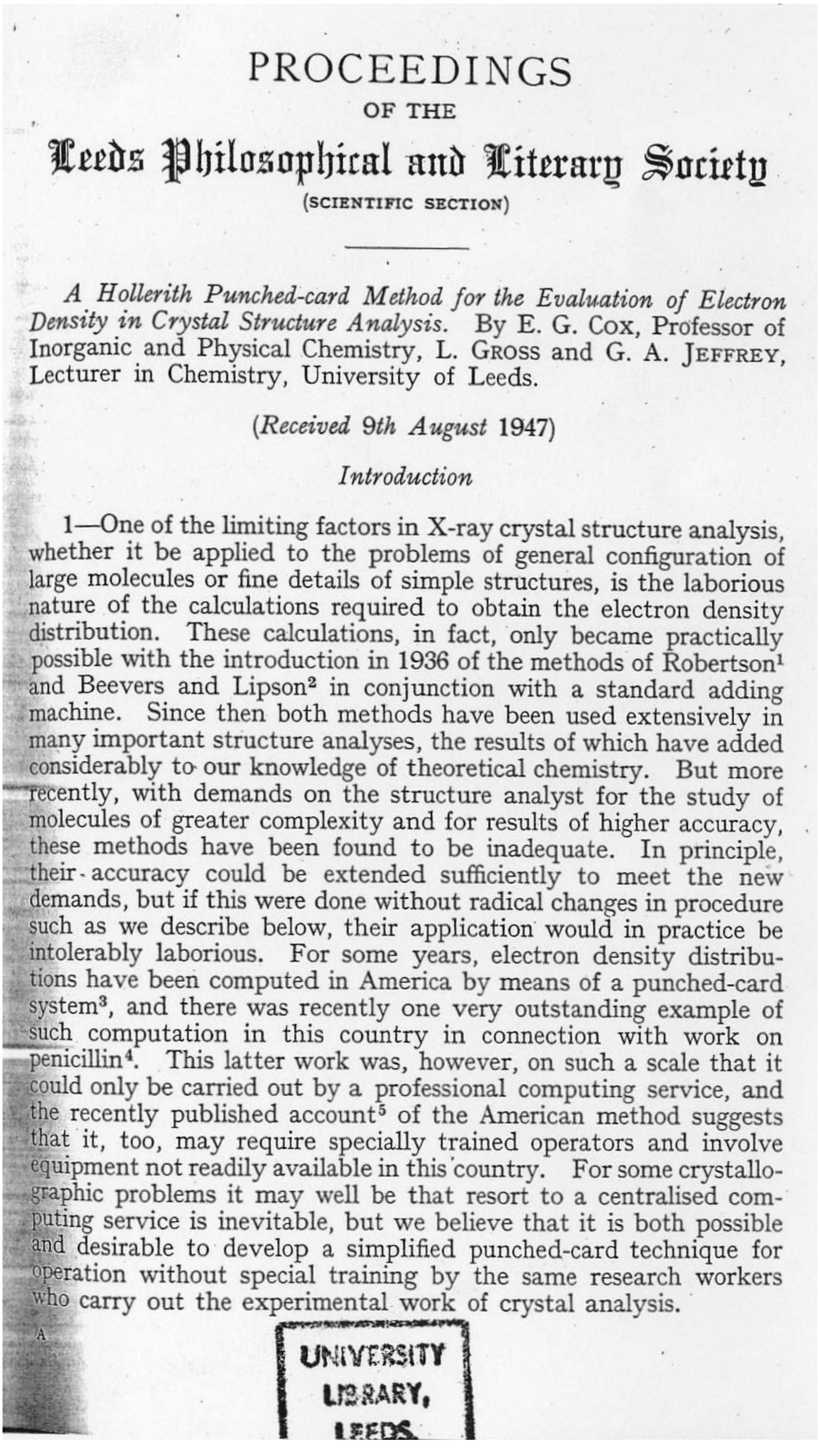
I joined Leeds University as a research (i.e. graduate) student in 1945. As I recall, the Department of Chemistry was subdivided into Organic, Inorganic & Physical. Prof. E.G. Cox was Head of Physical having come from Birmingham University and bringing G. A. Jeffrey as Lecturer with him. In Birmingham Cox had solved the structures of Werner complexes and Jeffrey had studied sugars. All matters of structural interest were pursued at Leeds. Although it was known in 1947 that the Americans were already using punched cards for Fourier summation, this was of little use to us. A locally-available, easily used method was needed.

The nearby West Riding Electricity Board ('riding' referred to an administrative third of Yorkshire county) used punched card equipment for accounting. Here they had tabulating machines which read cards and control boards whose plugs could be set for any particular summation. These boards could be kept for all summations of a given type.

At that time Fourier summation was largely two-dimensional and carried out with Beevers-Lipson strips and adding machines. Three-dimensional calculations would have been extremely arduous. Cards were therefore punched (by hand!) in a manner analogous to the strips so that, once reasonably good 3d phases were available, the electron density around each atom could be calculated.

The requisite cards obtained from the dispenser (loc. cit., Fig. 2) were stacked in boxes and taken, outside normal working hours, by tram (street car) to the Electricity Board. Hollerith-trained operators set the boards for us and the required columns of printed output taken away late at night. It is difficult to estimate how much faster all this was as compared to strips and adding machines. Five times faster? (We tried to make sure that the control boards needed for accounts were replaced for the next working days use. On one occasion this did not happen and the customers' accounts looked really weird!). Once back at the University, some hapless person had to resort the cards back into the dispenser.

I used this method to evaluate the atomic coordinates in diphenyl tetrafluroethane in 1949. It is not certain how long this method of calculation was continued at Leeds because those people who might know are no longer accessible. On leaving Leeds I had a long period in which either the required crystallographic calculations could be done by hand or by Beevers-Lipson strips. I did not invoke electronic computers as we now know them until using Deuce at the National Physical Laboratory in 1957.



PROCEEDINGS
OF THE
Leeds Philosophical and Literary Society
(SCIENTIFIC SECTION)

A Hollerith Punched-card Method for the Evaluation of Electron Density in Crystal Structure Analysis.

By E. G. Cox, Professor of Inorganic and Physical Chemistry, L. Gross and G. A. Jeffrey, Lecturer in Chemistry, University of Leeds.

(*Received 9th August* 1947)

Introduction

1—One of the limiting factors in X-ray crystal structure analysis, whether it be applied to the problems of general configuration of large molecules or fine details of simple structures, is the laborious nature of the calculations required to obtain the electron density distribution. These calculations, in fact, only became practically possible with the introduction in 1936 of the methods of Robertson[1] and Beevers and Lipson[2] in conjunction with a standard adding machine. Since then both methods have been used extensively in many important structure analyses, the results of which have added considerably to our knowledge of theoretical chemistry. But more recently, with demands on the structure analyst for the study of molecules of greater complexity and for results of higher accuracy, these methods have been found to be inadequate. In principle, their accuracy could be extended sufficiently to meet the new demands, but if this were done without radical changes in procedure such as we describe below, their application would in practice be intolerably laborious. For some years, electron density distributions have been computed in America by means of a punched-card system[3], and there was recently one very outstanding example of such computation in this country in connection with work on penicillin[4]. This latter work was, however, on such a scale that it could only be carried out by a professional computing service, and the recently published account[5] of the American method suggests that it, too, may require specially trained operators and involve equipment not readily available in this country. For some crystallographic problems it may well be that resort to a centralised computing service is inevitable, but we believe that it is both possible and desirable to develop a simplified punched-card technique for operation without special training by the same research workers who carry out the experimental work of crystal analysis.

A

UNIVERSITY
LIBRARY,
LEEDS

2—We have therefore evolved a method[9] which (i) makes use of the Hollerith equipment commonly available* in this country, and (ii) involves the manipulation of a comparatively small number of punched-cards (an important factor if the selection and replacement of the cards are to be carried out by one or two research workers not especially trained for that type of work). Mainly for these two reasons we have not attempted to develop a method for the complete three-dimensional computation of electron density throughout a whole unit cell, but rather to adopt the procedure of exploring the electron density over limited regions by means of sectional syntheses. It is expected that, at the stage of the analysis when the method is used, the positions of every atom will be known approximately and it will only be necessary to explore those parts of the unit cell in the vicinity of each atom. This is generally the method of employing the Beevers-Lipson strips for three-dimensional syntheses[6], but with the punched-cards it is possible in a much shorter time to compute the electron density at smaller intervals and with a higher degree of accuracy should the experimental data warrant it.

In general principles, the method is akin to that described recently by Shaffer, Schomaker and Pauling[5] although, in fact, it was evolved prior to receiving that information in this country. It is, however, different both in its object as indicated above and in the details of the punched-cards, and their manipulation.

General description of the method

3—Full descriptions of the methods of determining electron densities in crystals are available[8], and we shall therefore proceed directly to discuss the computational procedure.

The expression for the electron density at any point—

$$\varrho_{xyz} = \frac{1}{V} \sum_{-H}^{H} \sum_{-K}^{K} \sum_{-L}^{L} \left| F_{hkl} \right| \operatorname{Cos} 2\pi\left(\frac{hx}{a} + \frac{ky}{b} + \frac{lz}{c} - \alpha_{hkl}\right)$$

can be re-arranged as the sum of terms involving products of the sines and cosines of $2\pi\frac{hx}{a}$, $2\pi\frac{ky}{b}$, and $2\pi\frac{lz}{c}$. In general, for a non-centrosymmetric structure, there will be four such terms with coefficients $| F_{hkl} | \cos \alpha$ and the same number with $| F_{hkl} | \sin \alpha$. In a centrosymmetric structure this reduces to four terms only, and special crystal symmetry may further simplify the synthesis. If these component terms are evaluated separately and combined at the end of the calculations, then it is sufficient to evaluate for the

* The equipment required for the method as described here is (i) an 80-column Rolling-Total Tabulator, (ii) a Sorter, and (iii) a Comparing-Reproducer. (A full description of the Hollerith system, with examples of its applications, is given by Comrie, Hey and Hudson, *J. Roy. Stat. Soc.*, **4**, 210 (1937).)

points x, y and z, from the origin to half the cell dimensions. Furthermore, if the summations are separated in respect to odd and even orders of h, k and l, then only one quarter of the cell dimensions need to be considered.

4—The principle of the sectional "three-dimensional" synthesis is to fix a particular parameter, for example x, and carry out the summation of the observed F_{hkl}'s over all values of the appropriate index, in this instance h, thereby deriving a new set of coefficients, C_{kl}. These C_{kl}'s are then used in what is mathematically a two-dimensional synthesis, which in the past has usually been evaluated over all required values of the other two parameters (y and z) with the aid of strips tabulating A cos nt and A sin nt. In the Beevers-Lipson method these strips have values of A from $-$ 99 to + 99, and n from 0 to 20, any one strip (fixed A and n) having entered on it, to the nearest whole number, sixteen values of the sine or cosine function at 6° intervals of t from 0° to 90°; their use for two-dimensional projections of electron density are fully described elsewhere[2]. Our Hollerith punched-cards contain similar information but with amplitudes (A) of $\pm$ 1, $\pm$ 10, $\pm$ 100 only, (n) from 0 to 20 and the functions entered to 0·01 for 3° intervals of the argument t over the range 3° to 90°, i.e. for intervals of 1/120 of the cell-edge.

For any particular value of the amplitude A and the order n there are four Hollerith detail cards, one (a) tabulating values of A cos nt in 6° intervals from 6° to 90°, a second (b) giving values from 3° to 87° also in 6° intervals, and two more similarly for the sines. Unlike the Beevers-Lipson strips, on which corresponding positive and negative values are printed on two sides of the same strip, different cards are required for + A and $-$ A, and altogether there are 492 cards in a "master-pack", 252 cosines and 240 sines, there being no zero-order sines. In the following we refer to cards of type (a) as "even" and to type (b) as "odd" (because they contain values of the function for even and odd multiples, respectively, of the unit 3° argument). These must not be confused with odd and even *orders* (values of n).

5—Any three-figure amplitude is passed into the Hollerith adding mechanism (the Tabulator) by operating on cards of amplitudes 1, 10 and 100 with "digiting cards", which serve to transfer numbers from the first of a pair of counters to the second on which the totals are accumulated until the passage of a control card causes the total to be printed and the counter cleared for the next summation. This is best illustrated by an example. To add into the Tabulator the functions corresponding to an amplitude + 529, we put cards through the machine in the following order—

ix. $\left\{\begin{array}{l}\text{one digiting card,}\\ \text{detail card for } A = +1,\end{array}\right.$
viii. one digiting card,
vii. ,, ,, ,,
vi. ,, ,, ,,

v.	one digiting card; detail card for A = + 100,
iv.	one digiting card
iii.	,, ,, ,,
ii.	one digiting card; detail card for A = + 10
i.	one digiting card

Then during the passage of this "deck" of 12 cards the A = + 1 card is "rolled" from the first to the second counter nine times, the A = + 100 five times, the A = + 10 twice, giving the equivalent of a card of A = + 529.

One set of nine digiting cards is required for each summation, and these are conveniently spaced in the required relationship to the detail cards by first laying them out on a table and then placing the detail cards appropriately over them.

For example, in the addition of

+ 396	C.1
− 158	C.2
− 209	C.3
+ 138	C.4
+ 113	C.5
+ 444	C.6

the routine is as follows—

On the number

9 digiting card is laid	+ 10.C.1, − 1.C.3
8 ,, ,, ,, ,,	− 1.C.2, + 1.C.4
7 ,, ,, ,, ,,	no card
6 ,, ,, ,, ,,	+ 1.C.1
5 ,, ,, ,, ,,	− 10.C.2
4 ,, ,, ,, ,,	+ 100.C.6, + 10.C.6, + 1.C.6
3 ,, ,, ,, ,,	+ 100.C.1, + 10.C.4, + 1.C.5
2 ,, ,, ,, ,,	− 100.C.3
1 ,, ,, ,, ,,	− 100.C.2, +100.C.4, +100.C.5, +10.C.5

The cards are then picked up in sequence commencing with the number 9 digiting card and ending with the 10.C.5, forming a "deck" for insertion in the Tabulator. A "control card" is placed at the end of the deck to initiate the operation of printing the totals, which have been accumulated in the second of each pair of counters, and of clearing for the next summation.

It is evident that by this method of digiting, errors due to rounding-off the values of the functions to any particular decimal place are multiplied, in the worst case by 9, and may produce errors in the totals of a unit or so in the next decimal place. Taking a specific case from the example quoted above, the summation of the function for t = 12° by means of our Hollerith cards, on which entries are made to 0·01, is in error by 0·06; if the errors in the individual terms had happened to have been all in the same sense, the error in the summation would have been 0·09. In actual determination of electron density, summations may involve considerably more terms than the above example, and two successive summations are necessary, so that final errors of a few units in the first decimal place are likely. It is mainly for this reason that we enter the functions to 0·01 on the cards, so as to be quite sure that rounding-off errors cannot affect the accuracy to the left of the decimal point.

Details of the Punched Cards

6—Manilla cards, measuring $7\frac{3}{8}$ in. by $3\frac{1}{4}$ in., form the unit records upon which the functions A cos nt and A sin nt are punched in the form of small rectangular holes representing, by means of their vertical position on the cards, the digits 0 to 9. Since only one hole may be punched in each column (except for over-punching referred to below) and the card accommodates eighty columns, the maximum number of digits which can be represented on a single card of this type is eighty. The eighty columns can be divided arbitrarily into groups or "fields" for specific purposes; the first five columns are reserved for descriptive purposes while the remaining seventy-five are split into fifteen additive fields of five-figure numbers.

In the descriptive or indicative field, the following information is recorded (*see* Fig. 1)—

Column 1 shows the type of card: Hole in row 1, Detail card
Hole in row 5, Control card
Absence of hole, Digiting card.

Column 2 has a hole in row 1 for odd orders (n odd) and a hole in row 2 for even orders (n even).

Columns 3 and 4 give the actual order number which may have any value from 0 to 20 (1 to 20 in the case of sines).

Column 5 shows the amplitude by a hole in row 1 for unity,
in row 2 for ten, and
in row 3 for a hundred.

As further descriptive information is necessary, "over-punching" above the normal range of ten rows is used. In this way, negative values of A are indicated in column 5, cosines in column 6, and the odd series of 3° intervals in column 80. The absence of holes indicates respectively positive, sines and even series of intervals.

In the fifteen additive fields are punched the functions corresponding, on the even cards, to the values of t in 6° intervals from 6° to 90° and, on the odd cards, from 3° to 87°. The 0° values cannot conveniently be included, but, since the summation of such terms merely involves straightforward addition of the amplitudes, this is no great disadvantage.

Of the five columns in each field, four are used when registering the positive functions to the second decimal place, and five for giving the negative values in complement form. This is necessary because

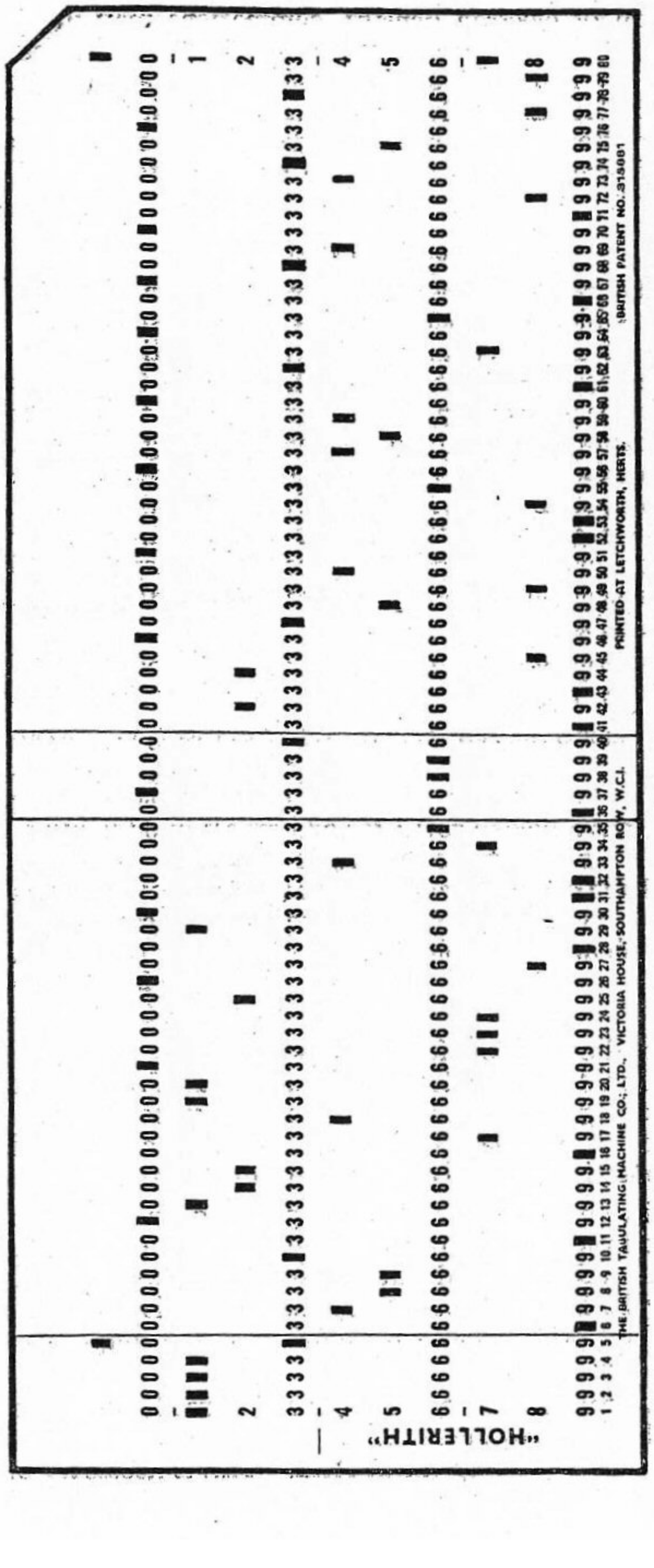

"HOLLERITH"
THE BRITISH TABULATING MACHINE CO., LTD., VICTORIA HOUSE, SOUTHAMPTON ROW, W.C.1.
PRINTED AT LETCHWORTH, HERTS.
BRITISH PATENT NO. 315081

Fig. 1

Example of a Hollerith card

Comparison with the description in the text will show that this card gives values of − 100 sin 11 t for odd intervals of t. For example, the holes in columns 36–40 read 906·63, i.e. − 93·36 which is − 100 sin 11.39°.

direct subtraction is not possible on the standard tabulator and must be effected by addition of the complement with respect to 999·99; e.g. − 56·42 becomes 943·57. The number + 100·00 is actually punched as + 99·99, introducing a small, and in this instance insignificant, error for the sake of economy in the use of adding wheels. (See remarks on accuracy in para. 5 above.)

7—The master-pack was prepared from a calculated table of the appropriate sines and cosines by means of a standard Hollerith hand-punch. It was verified by means of a similar type of machine and further checked by summing positive and negative cards on the tabulator, when each additive field gave a zero total. It could then be reproduced any number of times at the rate of 100 cards a minute by means of a Comparing Reproducer, which machine also verified that the duplicate packs were correct.

It is convenient to have 30 duplicates of all cards, except for orders above 10 where it is sufficient to have ten duplicates of the cards with amplitudes ± 100. The total of some 14,000 cards were arranged in 492 sets of like cards by means of a Hollerith Sorter operating on the information in the indicative field of each card, prior to being placed in pigeon-holes for selection as required for the Fourier synthesis. Details of this pigeon-hole system are a matter of individual choice, but the following suggestions are based on our experience—

(i) Since odd and even, and sine and cosine cards are never mixed in any synthesis, they should be filed in four separate systems. It may be noted that Hollerith cards lend themselves very readily to visual inspection by categories. For example, all cosine cards are over-punched in column 6 so that it is possible to see a clear hole in this position through any pack of cosine cards; thus if a sine card is accidentally included in a cosine pack its presence can be immediately detected visually.

(ii) It must be possible to mount the table of coefficients showing the cards required in close proximity to the pigeon-holes, and it is desirable also to arrange that the coefficient and the pigeon-holes corresponding to each order are indicated automatically in turn. There will be six such pigeon-holes for any one class of card and an arrangement mounted on a revolving table as indicated schematically in Fig. 2 appears to be the most convenient.

The Procedure for a Three-dimensional Sectional Synthesis

8—As in the evaluation of a two-dimensional projection[2], the calculations are carried out in two stages; summing first for values of one parameter to give coefficients in a preliminary table and then using these coefficients in summing for the second parameter to give the electron densities at all the required points in the section through the unit cell.

Supposing that the crystal symmetry is such that the general expression for the electron density reduces to a four-part synthesis, then eight packs of cards must be assembled, hand-picked in the way indicated above to correspond to the initial tables of coefficients, the number of "decks" (paragraph 5) in each pack being equal to the number of columns in the initial table. From each table of

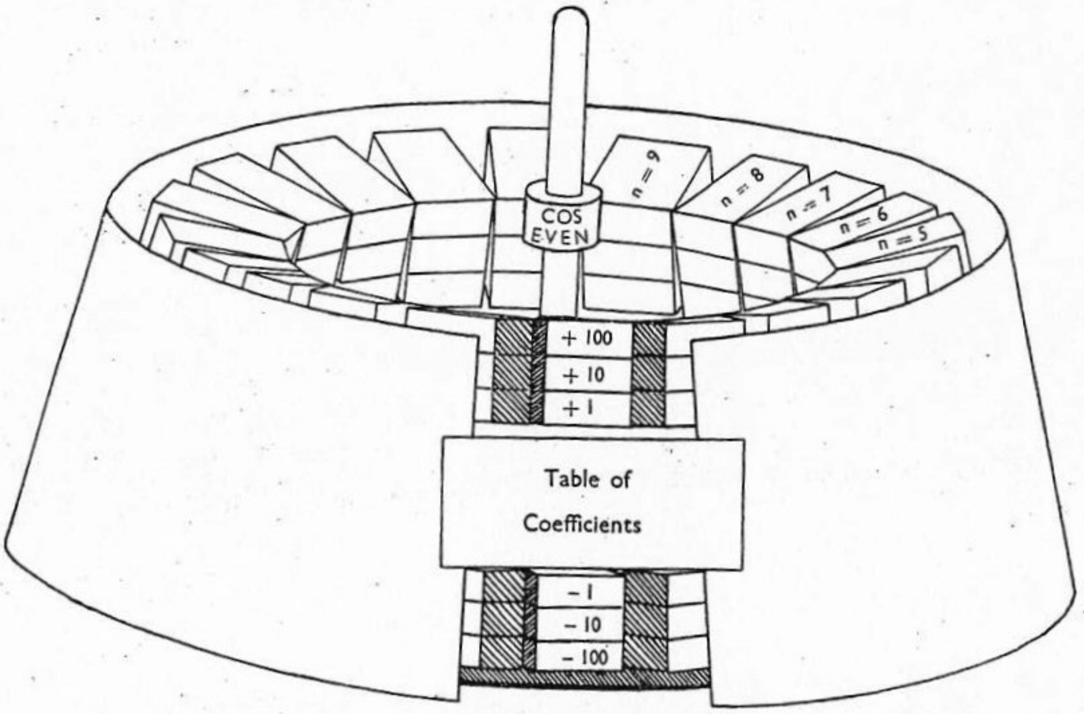


FIG. 2
Schematic representation of the card selection system

coefficients there will be two packs, one of odd cards and the other of even cards. These two packs should be identical in respect of both the arrangement of the various types of cards (i.e. sine or cosine detail cards, digiting and control cards) and the descriptive punching of the amplitude and order numbers. This fact offers an excellent means of checking for mistakes in the selection of cards, as the two packs can be compared by passing them simultaneously through a Comparing Reproducer. This machine allows the cards to pass through continuously (at the rate of 100 per minute) only if pairs of cards taken in order from the two packs are identical in respect of the holes in certain pre-selected columns; otherwise the machine will stop, indicating a mistake in one or other of the packs. Reference to the description of the indicative information on the cards will show that the Comparing Reproducer must be set to operate* on the normal punching in columns 1, 3, 4, and 5, and the over-punching in columns 5 and 6. As a further safeguard against systematic errors, it is advised that the corresponding odd and even packs be assembled by different operators, when the chances of getting identical mistakes in both packs become negligibly small.

The eight packs are then ready for insertion in any desired order through the Tabulator, which will print totals at the receipt of each control card, i.e. at each individual summation, and for any predetermined set of additive fields (*see* below)*.

Since the standard Tabulator has only six counters which, for the purposes of this method, are coupled into three pairs, it can only add the numbers in three fields of the cards during the one run through the machine. This means that any particular pack has to be put through the tabulator once for every three values of x/a, y/b and z/c that are required, and is the reason why the method is more applicable to syntheses over narrow limits than throughout a whole unit cell. It is suggested that in the immediate vicinity of the maximum of an atomic peak the electron density should be evaluated at 3° (1/120th) intervals, and that for outlying contours 6° (1/60th) intervals should suffice. The number of points considered will depend upon the particular cell dimensions; for an axis length of about 15A., six points at 1/120ths flanked by three points on each side at 1/60ths should be adequate for most purposes. This will entail putting the eight packs through the Tabulator once only, followed by two separate runs of four packs.

9—By suitable manipulation of the paper the totals from the first summations can be printed in a form convenient for the selection of the cards for the summation over the second variable parameter. Again there will be eight packs of cards, and in this case the four totals corresponding to any one particular 1/120th, when added with appropriate signs, give the electron density values at the point. By adjustment of the printing mechanism, the four totals can be printed in suitable positions for convenient addition. If desired, they can also be permanently recorded on Hollerith cards, for subsequent addition or reference, by coupling the Tabulator to a Reproducer Summary Card-Punch (a special type of Comparing-Reproducer).

10—A modification of this procedure becomes necessary if the range of one or both the variable parameters passes through the 30/120 value. It is then required to subdivide the packs further and evaluate for odd and even orders separately, when the sum of the ensuing totals will give values for points such as 25, 26, 27, 28, 29 and 30 1/120ths, while the differences give 31, 32, 33, 34 and 35 1/120ths. It will be appreciated that although this involves making up twice as many separate packs, there are not twice as many cards to pass through the Tabulator, and generally time lost in taking the extra totals will be more than gained by the greater range covered in a single run of the cards through the machine.

* The plugging of the control boards of the machines for the required operations must be done by a Hollerith-trained operator; it should be noted, however, that a control board can be withdrawn fully plugged and stored for subsequent use while the Tabulator is used for other purposes.

11—There remains only to re-assemble the cards in their filing system. The sines and cosines, odds and evens remain in their separate packs throughout the whole procedure. The remaining classifications are sorted in the following stages—

(1) type of card; detail, control and digiting (column 1)
(2) amplitudes; positive and negative (column 5 over-punching)

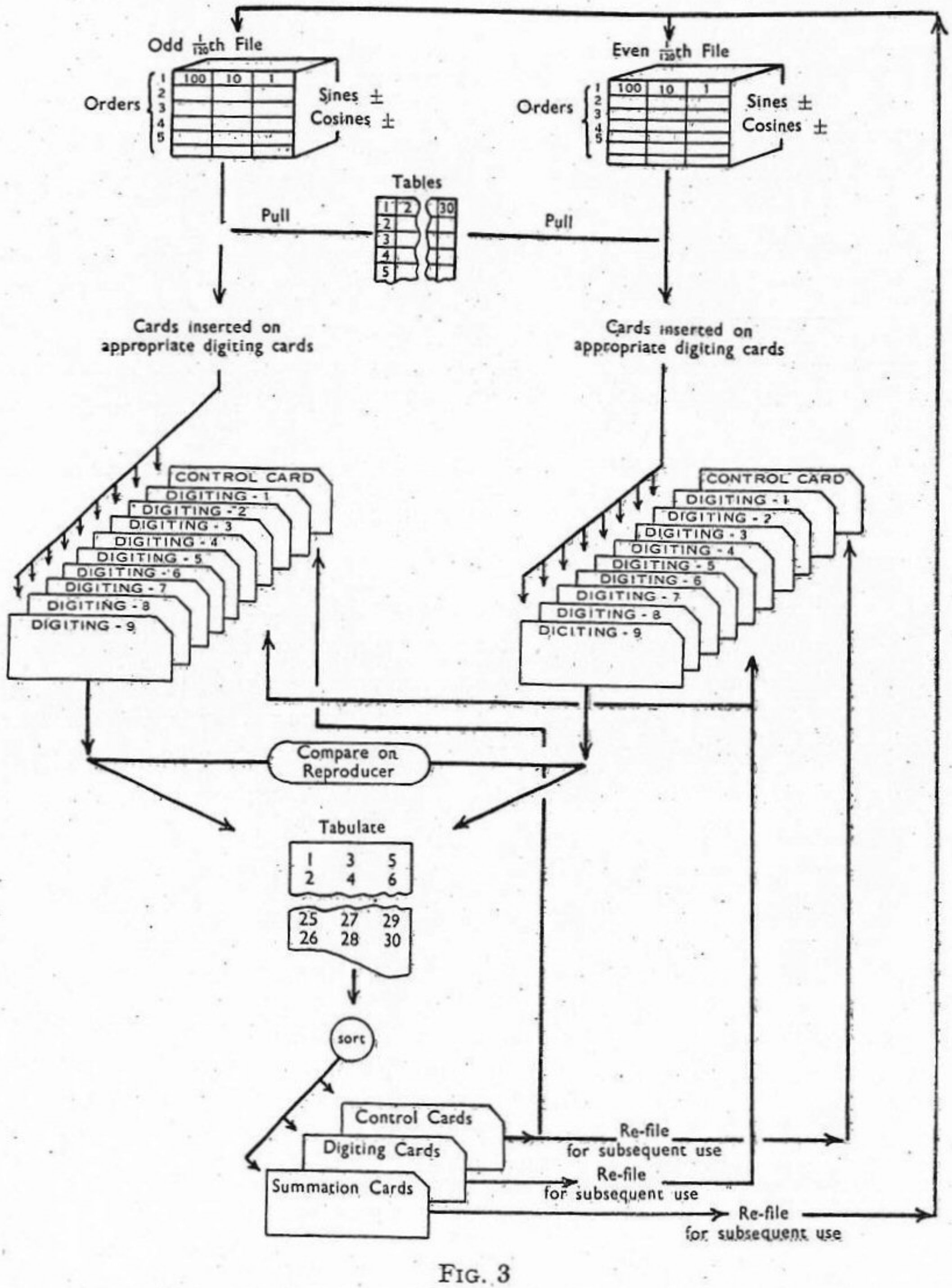


FIG. 3
"Flow diagram" for the evaluation of electron densities by the punched-card method

(3) amplitudes; units, tens and hundreds (column 5)
(4) order numbers; 0 to 20 (columns 3 and 4).

The cards are then arranged in a suitable order for systematic replacement in the pigeon-holes, and rapid inspection through the punched holes ensures that all the cards in any one pigeon-hole are identical.

The whole routine for the calculations is indicated schematically in Fig. 3.

Operational Times

12—As a guide to the time for the evaluation of a sectional synthesis by this method details are given below for a particular example. The structure analysis concerned was that of dibenzyl[7], for which the cell dimensions are a = 12·77A, b = 6·12A, c = 7·70A, $\beta = 116°$, and maximum values of h, k and l are 14, 7 and 8, respectively. A sectional synthesis was calculated at a y value of 0·096 to explore an atomic peak in the vicinity of x, z equal to 0·026, 0·975.

The first stage of the summation was at particular values of h and all values of l. The structure factors used were expressed to the first decimal place, and as a result, in each of the four initial tables there were roughly 30 three-figure numbers, 70 two-figure numbers and 20 one-figure numbers. Using a provisional filing system for the Hollerith cards, the time for the selection of cards was 15 minutes per pack. With two operators pulling the corresponding odd and even cards, the total time for the complete set of eight packs was one hour. It is not expected that the more complicated pigeon-hole system suggested above will shorten this time, but in reducing mistakes it will also reduce the number of stops and corrections which are made during the comparison of the odd and even packs.

The running-time through the Comparing Reproducer with no stops for mistakes was 5 minutes per pair of packs; 20 minutes altogether. But this time is not so important because as soon as the first pair of packs have been checked they can be put on to the Tabulator without waiting for the other comparisons.

The Tabulator was arranged to give totals from the fields corresponding to z values of 1, 2, 3, 4, 5, and 6 1/120ths in the first run of the eight packs. This was followed by a run of the even packs only for the values of 8, 10 and 12 1/120ths. The totals were printed at a rate of six (i.e. 1/120 to 6/120) for each value of h every 35 seconds. The time for a pair of odd and even packs and all 14 values of h was 9 minutes. The complete first-stage summation involving four sets of coefficients at the nine z values was completed in one hour.

The cards were then sorted for replacement in the filing system at the rate of 6 minutes per pack, all eight packs being completed within one hour.

The process was repeated for the second stage of the synthesis using the new coefficients and evaluating at the nine particular values of z and values of x at 1, 2, 3, 4, 5, 6, 8, 10 and 12 1/120ths. The operating times were similar to those for the preliminary table.

The final addition of the four components of the electron density with signs appropriate to the particular quadrant of the unit cell was done directly from the printed sheets with an ordinary adding machine.

The replacement of all cards into their pigeon-holes ready for the next synthesis was left until the end, and took rather less time than the original selection.

Summarising, with two (untrained) operators and an improvised pigeon-hole system, the times were—

for selecting cards,	2 hours
tabulating,	2 hours
sorting,	2 hours
replacing,	ca. $1\frac{1}{2}$ hours
Total time,	ca. $7\frac{1}{2}$ hours

It is difficult to determine the comparable time using the Beevers-Lipson method to the same accuracy since the necessary strips are not available, but this Hollerith method is certainly much faster, and less onerous for the operators.

Comparing the actual positions of this particular atomic peak, the results obtained from this synthesis differ from a Beevers-Lipson strip synthesis at 1/60ths by 0·005 A in the direction of the a-axis and 0·002 A in the direction of the c-axis. These differences arise in the first place from rounding off errors in the Beevers-Lipson strips, and secondly, from the more accurate peak-location with intervals of 1/120th. (A paper giving a full discussion of computational and other errors in Fourier syntheses is now in preparation in this Department.)

Acknowledgments

13—The development of this method has been greatly assisted by Mr W. McL. Wishart and Mr J. L. Ineson of the Central Electricity Board, who have placed Hollerith machines at our disposal. We have also to thank Mr J. Grant, of the British Tabulating Machine Co., for his helpful interest, and Messrs. Bartindale, Cruickshank, Gillot, Nyburg and Stadler for help in carrying out trials.

Summary

14—This paper describes a method of using punched cards and Hollerith equipment for the evaluation of electron densities in crystals by three-dimensional Fourier synthesis of X-ray data. The method is fast and accurate and can be used by research workers not specially trained on Hollerith equipment.

It is intended for the evaluation of electron density over comparatively small areas to locate accurately atoms whose approximate positions have already been determined by less precise methods.

References

[1] Robertson, J. M. *Phil. Mag.*, **21**, 176 (1936).
[2] Lipson, H., and Beevers, C. A. *Proc. Phys. Soc.*, **48**, 772 (1936).
[3] Hughes, E. W. *J.A.C.S.*, **63**, 1737 (1941).
Chia-Si-Lu, Hughes, E. W., and Giguere, P. A. *J.A.C.S.*, **63**, 1507 (1941).
Levy, H. A., and Corey, R. B. *J.A.C.S.*, **63**, 2095 (1941).
Waser, J., and Chia-Si-Lu. *J.A.C.S.*, **66**, 2035 (1944).
[4] *Nature*, **156**, 766 (1945).
[5] Shaffer, P. A., Schomaker, V., and Pauling, L. *J. Chem. Phys.*, **14**, 648 (1946).
[6] Goodwin, T. H., and Hardy, R. *Phil. Mag.*, **25**, 1096 (1938).
[7] Jeffrey, G. A. *Proc. Roy. Soc.*, A.**188**, 222 (1947).
[8] e.g. Robertson, J. M. *Reports on Progress in Physics*, **4**, 332 (1938).
[9] Already briefly described in *Nature*, **159**, 433 (1947).

Chemistry Department
The University, Leeds

Some Powder Indexing News

Jon Wright
European Synchrotron Radiation Facility, BP-220, Grenoble, 38043, France. E-mail: wright@esrf.fr ; WWW: http://www.esrf.fr/exp_facilities/ID11/handbook/staff/jon/jon.html

In the last issue of this newsletter [1] Robin Shirley gave an overview of the powder indexing problem, an area which has seen something of a renaissance in the last few years. Robin anticipated some more developments to be found on the conference trail this summer and therefore this article is not meant to be an exhaustive review, but more of a sharing of notes from those talks I was able to attend this year.

Before immediately getting to the state of the art it is perhaps wise to remember where things stood with indexing unit cells from powder data and where the difficulties might lie. Obtaining a unit cell from a powdered sample requires not only a sample and a dataset but also a human operator to collect and 'interpret' the data, and frequently nowadays a computer program to aid the human's thinking.

The main thrust of the recent efforts in the field has been the development of improved software, although it rarely seems to be the case that the software is the problem. These new programs aim to improve upon those which were previously available, which is something of a Herculean task, given the high success rate of programs like TREOR [2], ITO [3] and DICVOL [4] (etc, with apologies to those not listed), particularly when those programs are used in combination. For well crystalline materials which do not have a unit cell that is considered 'difficult', those auto-indexing programs will usually yield the correct unit cell very quickly; provided that a good quality dataset is available. A 'difficult' unit cell is usually one where a single axis is very short compared to the other two, giving rise to a dominant zone. The first 20 (or more) lines might all index as (hk0) making it hard to find the three-dimensional unit cell. Lines which should be indexed as (hkl) are only found in the high angle region of the pattern and may be overlapped. There are also materials such as quasicrystals which will continue to frustrate the indexing programs, even if sharp diffraction patterns can be obtained.

Problems with the dataset can also prevent the algorithms from finding any correct solutions. This can be due to small misalignments (zero shift, specimen displacement) or more fundamental problems with the experiment (the first 40 lines were blocked by the beamstop…). The most difficult problems are those which are presented by the sample. There might not be as many sharp diffraction lines as you had hoped for, and frequently samples are mixtures of several crystalline phases, meaning that more than one unit cell needs to be identified. For many researchers, it is probably fair to say that impurities cause the most problems for finding unit cells and structures for new materials.

A wide variety of 'operator errors' remains one of the hardest aspects of indexing to circumvent, but this is not related to the algorithms being used within the indexing programs. An integrated software package which includes automatic peak search, peak fitting, unit cell determination, pattern decomposition and identification of the extinction symbol goes a long way to removing the opportunities for human creativity! There is always a price for progress.

Having introduced the problem it is time to talk about my travels this year. I managed to survive the local hospitality in Durban and arrived intact for the start of the indexing session at the ECM meeting. The session was chaired by Robin Shirley and C. Rademeyer. The first talk was given by Robin Shirley who tackled one of the key problem in indexing; deciding whether a proposed unit cell is correct, or whether it is one of a multitude of larger unit cells which coincidentally index the observed lines, as well as generating many which are not observed. With multiphase samples, where impurity lines must be tolerated, the conventional figures of merit frequently fail to identify the correct unit cell. Robin described scoring methods based on joint probability distributions and was able to show that the correct cell remains the highest peak in a section of parameter space even when what seemed like an overwhelming multitude of impurity lines were introduced. These improved figures of merit offer great promise for the difficult problems which are now being tackled and should be available in the Hmap part of the forthcoming release of the crysfire suite [5] which Robin maintains.

Vicktor Zlokazov has confronted the multiphase indexing problem and presented the second talk of the session. The AUTOX program uses genetic algorithms to find complementary unit cells for multiphase datasets. By ensuring that the 'left over' lines in a powder pattern canalso be indexed the potential for incorrect proposed unit cells is drastically reduced. Examples with higher symmetry unit cells were presented, where the program quickly finds the right answers.

Another recent development in the field is the McMaille [6] program, written by Armel Lebail who gave the third talk. This program uses a figure of merit which is based on the quality of fit which a proposed unit cell will give in a pattern decomposition (the Rp for a Lebail fit), with approximations to make the program faster. By using this figure of merit the program is naturally tolerant of weak impurity lines and makes a significant step in that the figure of merit is now weighted by the intensities of the observed lines. A grid search algorithm allows the user to search parameter space exhaustively, with a Monte-Carlo based algorithm available for finding the right answer more quickly. In a series of test cases McMaille was shown to find the right unit cells for patterns containing progressively more and more spurious lines, it appears to tolerate up to ~15% impurity, although the figures of merit are reduced as the amount of impurity increases. This program will be a very welcome development for the practical synthetic chemist or materials scientist who deals routinely with impure samples. Being an open-source effort, there is also an opportunity for programmers with new ideas to contribute and improve the program further.

One of the factors mentioned earlier was the presence of 'operator error' in the process of determining the unit cell. In the fourth presentation in this session at ECM the SVD index [7] part of the TOPAS package was described. Unfortunately Alan Coehlo (who wrote the program) was not in Durban, so it was not until I caught up with him at the SDPD workshop in Stara Lesna that a more complete story emerged. Alan insists the idea is very simple, just carrying out unit cell refinements for many trial unit cells. Clearly there are a number of details which make the program more robust and efficient and it is in the handling of details where this software package excels. Possible unit cells are assigned appropriate extinction symbols and extremely useful plots characterising the cells tested can also be generated. Zero-shift refinement is included as a part of the parameter space so that the program can cope non-ideal data. Being commercial software the program is very robust and has been tested against a large set of problems. The algorithm is said to be 'semi-exhaustive' in as much as it has always found the answer where an answer exists. The question of whether or not an answer exists remains difficult, with all of the programs struggling when the data do not really distinguish between many different unit cells which all have low figures of merit. Plots of figure of merit versus zero shift for a large number of refined unit cells are very helpful for the task of identifying the correct cell (or deciding that you need better data). The package also includes an 'LP search' algorithm which uses the whole diffraction profile.

Also in Slovakia, Daniel Louër described some developments which have been going on within the DICVOL program. This has been something of a killer application for higher symmetry unit cells in recent years. Perhaps motivated by the XCELL program [8] from Marcus Neumann at Accelerys (which also uses a successive dichotomy algorithm) some key enhancements have been made in DICVOL. The program is now able to tolerate impurity lines and small zero shifts, which had always caused frustration for users. The program has also improved the handling of equivalent unit cells, recognising when it has found the same solution more than once. We look forwards to a new release once the program is fully tested. From my personal point of view, Daniel presented a far better way of dealing with impurity lines when he showed data collected for a pharmaceutical material as a function of temperature. Since the main phase and the impurity decompose at different temperatures it was clear which lines belonged to the impurity phase when they disappeared from the diffraction pattern! This experimental method looks to be a far better use of time than struggling to solve the problem with a computer.

Hopefully I have at least mentioned most of the main developments in the area seen on this summers travels. With all of these new developments, one might feel a little daunted as to what is the accepted best practice to be tried before admitting that a particular powder pattern is un-indexed. Is it worth buying a commercial program? Are you exploiting all of the features available in the programs which are available? Would it be worth trying to get improved data before spending too much time at the computer? Some guidance might be found at Armel Le Bail's web page, where the results of the "Unindexed Powder Pattern of the Week" challenges can be found [9]. During autumn 2003 Armel proposed a series of problem on the SDPD mailing list where software authors and users could try out their software. I mention this here in the hope that more potential competitors who might have missed the announcements can participate in the event. With one problem per week the programs are going to have to be very efficient and user friendly to allow people to keep up! Consistently promising proposed unit cells have come from TOPAS/SVD Index, Crysfire, McMaille and AUTOX, suggesting that all of these programs are doing excellent jobs at the cutting edge.

As closing remarks it only remains to re-iterate the instructions present with all good indexing programs: there is no replacement for good quality data. While it might be entertaining to watch the experts doing gymnastics to see who can find the best cell for the worst possible dataset, the average researcher remains better off with the best possible data set. On that note I look forwards to seeing developments in the future which can exploit more of the information available from a powder sample. Peaks can be characterised by their intensity and width, as well as their position, yet most programs only use the peak position information. When data can be collected under different conditions, then any anisotropic changes could be exploited in the determination of cell parameters. The use of texture has been proposed by Bunge and Park [10] as a way improving the criterion for determining whether a unit cell is correct or not, by exploiting the information pole figures which is related to the reciprocal lattice vectors. Another method, from Ferraris and Pavese [11], uses any anisotropic effect which modifies the lattice to identify central rows (delta_D/D is constant). This might be chemical substitution, temperature, pressure etc. It would appear that the software for deriving unit cells is now very close to being optimal in the sense that exhaustive searches can be made in a reasonable amount of computing time. Further improvements might be found where further experiments can be carried out and it is that area where there seems to be the most potential for further developments.

Acknowledgements

I'd like to thank the authors whose work I have mentioned here and hope they will forgive me for any errors and omissions.

[1] R. Shirley, *Comp. Comm. Newsletter* (2003) **2** 48.
[2] TREOR. P. E. Werner, L. Eriksson and M. Westdahl, *J. Appl. Cryst.* (1985) **18** 367.
[3] ITO. J. W. Visser, *J. Appl. Cryst.* (1969) **2** 89.
[4] DICVOL. A. Boultif and D. Louer, *J. Appl. Cryst.* (1991) **24** 987.
[5] Crysfire http://www.ccp14.ac.uk/ccp/web-mirrors/crys-r-shirley/
[6] McMaille http://www.cristal.org/McMaille
[7] SVD Index. A. A. Coehlo, *J. Appl. Cryst.* (2003) **36** 86.
[8] XCell. M. A. Neumann, *J. Appl. Cryst.* (2003) **36** 356.
[9] UPPW. http://sdpd.unic-lemans.fr/uppw/list.html
[10] H. J. Bunge and N. J. Park, *Materials Science Forum* (1996) **228-231** 25.
[11] G. Ferraris and A. Pavese, *Crystallography Reports* (1999) **44** 734.

Crystallographic Computing Commission Logo competition
(Submissions due by 2nd July 2004)

IUCr
Computing
Commission

Can you do better than this?

The IUCr Commission on Crystallographic Computing is holding a Logo competition and requests submissions to replace it's current rather moribound and lame Newsletter logo, and very plain webpage banner (both created by this humble scribe). Some might frown upon this emphasis on superficial eye candy, perhaps quoting John Lyly (1554-1606) "The sun shineth upon the dunghill, and is not corrupted". Irrespective, it would be nice to have some graphics that the sun felt happy to irradiate. Potential contributors should feel empowered to use any graphics they feel to be relevant to historical and modern computing and crystallographic symbols: e.g.,

punch cards, digital circuits, mathematical equations, bits of source-code, computer valves, fourier maps, B-L strips, coffee-mug stains, discarded chip/crisp packets, empty softdrink/pop cans, etc,

There are three different images that are requested of crystallographic artists:

- The IUCr Commission on Crystallographic Computing Newsletter Logo (currently a quickly drawn round thing with some text in it) http://www.iucr.org/iucr-top/comm/ccom/newsletters/
- A graphical banner logo for its webpages at: http://www.iucr.org/iucr-top/comm/ccom/ (final version for web will most likely be 500 x 120 pixels)
- A web banner logo for its 2005 Computing School being held before the Florence IUCr Congress at Siena (18th to 23rd August 2005) http://www.iucr.org/iucr-top/comm/ccom/siena2005/ (Venue: Certosa di Pontignano, University of Siena: http://www.unisi.it/servizi/certosa/ ; University of Siena webpage: http://www.unisi.it/)

NB: CompComm is the current abbreviation for the Commission on Crystallographic Computing

The IUCr Commission on Powder Diffraction logo may give people an idea of what has been done in the past for IUCr based newsletters: http://www.mpi-stuttgart.mpg.de/cpd/html/newsletter29.html
Or such things as the Shelx homepage may serve as a guide for a banner logo: http://shelx.uni-ac.gwdg.de/SHELX/

Submissions should be sent by 2nd July 2004 to Lachlan Cranswick at lachlan.cranswick@nrc.gc.ca . Preferably in a vector based format that is rescalable to small and large formats - and a JPG or GIF file representation. It would be nice if artists could contribute some explanatory text of the type given at the following "about the logo" pages. (a paragraph would be fine - and this is optional) http://www.cins.ca/cpdw/logo.html http://www.cins.ca/aca/ikaite.html

Artwork should be completely original and not using any copyright or trademarked material from other sources. All submissions become the property of the IUCr Computing Commission; and all submissions will be placed on its website for public view.

Lachlan M. D. Cranswick, NRC, Chalk River, Ontario, Canada (*Lachlan.cranswick@nrc.gc.ca*)

Meeting Reports relating to Crystallographic Computing

During ACA 2003, 26th to 31st July, Cincinnati, USA: SP02: Future strategies for successful crystallographic computing

Fig 1: *Left to right: Paul Adams, Sue Byram, Ross Angel, Brian Toby, David Watkin, Lachlan Cranswick, Carroll Johnson, Joe Pflugrath and Mathias Meyer.*

It was standing room only throughout the half-day session concerned with the future support of crystallographic software. The first half of the morning was devoted mostly to defining the issues, both old and new. **David Watkin** (Univ. of Oxford) opened the session with a review of the history of crystallographic software development. He particularly contrasted a golden past in which software diversity was maintained by widespread programming skills, public domain distribution, free maintenance, and open cooperation between developers with the gloomy present of market forces not only in commerce but also within universities and government labs. He noted examples of crystallographic software being effectively killed by universities demanding licensing fees and expressed a concern about "creeping entrapment" whereby free software is not maintained because of lack of funding, with the result that the underlying ideas are eventually lost to the community.

The new issue of software patents was reviewed by **Lachlan Cranswick** (Chalk River). He showed that patents on basic concepts such as reflection centering, structure solution and least-squares refinement are now being passed by patent offices leaving the entire community open to legal action. He summarized the potential dangers of such patents and patent actions with the memorable phrase that "we cannot stand on the shoulders of giants if the giants wear spiked shoulder pads." Readers interested in seeing whether they already violate a software patent ***(and that really does mean every single member of the ACA)*** should consult an article at http://www.iucr.org/iucr-top/comm/ccom/newsletters/2003jan/ and read Lachlan's review at http://www.ccp14.ac.uk/maths/software-patents/ . Some of the subsequent discussion focused on the role of GNU software licenses although it was clear that such licenses address issues of copyrights rather than patents. **Carroll Johnson** (Oak Ridge) continued the list of difficulties facing the modern programmer with a synopsis of security issues and the inadequate tools available to address them. A show of hands in the discussion session revealed the seriousness of the situation in that most present knew that their labs had been the victims of hackers and/or viruses, or believed that they may have been and have not yet found out.

Software development and diversity were the themes of the second half of the session. **Brian Toby** of NIST reflected on the life and career of a typical software developer in the US government labs where much of our current crystallographic software started life, but where such development is not directly supported. He felt that software and algorithm developers needed more public recognition by the community, a point that was well-received by the packed audience. And he used the 20 year development history of Rietveld codes as an argument to show that were software to be only developed on a purely commercial basis it would stagnate, as development timescales are far longer than is acceptable for commercial products. **Sue Byram** described strategies at Bruker/Nonius to ensure diversity by pursuing in-house software development in parallel with the integration of software from various other sources into their packages. Oxford Diffraction's **Mathias Meyer** described a slightly different solution to integration of commercial and academic code through offering open-source software and the integration of user's software packages through plug-ins.

Another way of promoting diversity in software is to make it easier for people to program algorithms by providing software "toolboxes." **Paul Adams** (Lawrence Berkeley National Lab) described the development of one such system, *Phenix*, and its associated cctbx toolbox. Subsequent discussion of the project focused on how the development of *Phenix* is supported financially, and whether it would survive in the longer term. Discussion got quite heated over the difference in cost between academic and commercial licenses. David Watkin meanwhile pointed out that a similar toolbox, the Cambridge Crystallographic Subroutine Library (CCSL) had been developed with the same aims more than 20 years ago. A show of hands indicated that only 3 of the 100 or so attendees at the session had heard of it, illustrating that long-term support of such resources for the benefit of the community can clearly be problematic. That show of hands also indicated to your surprised correspondent that he had joined the "older generation"!

The extensive discussions and thought-provoking presentations through the morning boiled these various issues down to software diversity, software support, and the training of programmers. Reassuringly, several speakers and discussion participants felt that there was no shortage of crystallographic programmers and that various methods are available to ensure the continuing development of that work force. Support of software development and maintenance and retention of software diversity are clearly inextricably linked as, in the words of the title of **Joe Pflugrath**'s presentation, "There is no such thing as free software". While there is general agreement that software diversity needs to be maintained, there are clearly differing views as to how this should be ensured or whether a simple evolutionary approach of "survival of the fittest" is to be desired. Distilling the views expressed in the general discussion session, your correspondent believes that a diversity of support mechanisms will ensure a diversity of software. Such support should range from grant-funded software archives and clearing houses such as CCP14, payment of realistic fees for commercial software, and the recognition that grant support is appropriate for both developers in the academic environment and to end-users to enable them to purchase commercial software. And most importantly, as George Sheldrick reminded everyone, the publishing of software notices and descriptions in journals such as *J. Appl. Cryst.*, and the citing of such sources by all those who use the software is essential. With such citations, programmers can justify their existence to supervisors and department chairs, as well as stand a chance of gaining financial support for their efforts from funding agencies. So, everyone left the session more reassured than at the start of the morning, but conscious that much needs to be done to raise the status of crystallographic programming back to where it was several decades ago. After all, without software we would all be using Beevers-Lipson strips!

Ross Angel

Crystallography Laboratory 3076 Derring Hall, Virgina Tech, Blacksburg, Virginia, 24060, USA, E-mail: rangel@vt.edu ; WWW: http://www.crystal.vt.edu/crystal/

During ACA 2003, 26th to 31st July, Cincinnati, USA: A Workshop on Twinning & CRYSTALS,

The aim of this workshop was to cover theoretical background to the phenomenon of twinned crystals, practical aspects of identification and treatment of twinned data sets, and some hands-on examples to work through using the PC's and software provided. The hands-on session also enabled participants to evaluate the freely available CRYSTALS software package as an alternative to the SHELX suite of programs.

Instructors: Simon Parsons, Victor Young, Richard Cooper. The workshop was attended by approximately 50 delegates.

Fig 1.: *Similar but not identical photos: Some of the workshop presenters - David Watkin, Simon Parsons, Victor Young and Richard Cooper* (images added by the newsletter editor)

The programme began with a basic introduction to the CRYSTALS software in order to familiarise everyone with the package. In the first session of the morning, Richard gave an overview of the key concepts and facilities of CRYSTALS and then a demonstration of a simple structure solution, refinement and analysis using CRYSTALS. This was followed by a hands-on session with up to three example structures for everyone to solve and refine.

CRYSTALS has been developed to make routine crystallographic structure determination a realistic prospect for non-crystallographers, by guiding the analysis and spotting problems as they arise. Recent developments for analysing data before and during the refinement were highlighted. Plots of I and σ(I), merging-R and systematic absence violations provide the chance to spot problems with data before even attempting to solve the structure. Later, plots of residuals, weighting schemes and F_o against F_c, prove to be very useful for spotting trends such as extinction and outliers in the data.

After looking at these routine structures, participants were invited to solve and refine a poor-quality data set again using the guided mode of the program, but making use of some graphical analyses to spot and correct for problems in the data.

After coffee the programme focussed on twinning. Simon Parsons gave a lecture on the topic of *merohedral* twins starting from a consideration of the metric symmetry of the unit cell and considering the effect of multiple domains in different orientations on the diffraction pattern. This led into the concept of the *twin law* for relating overlapping *hkl* indices. Most practical problems with these twins occur during space group assignment and structure solution, if these can be overcome, refinement is fairly trivial.

Simon handed over to Victor Young who concentrated on *non-merohedral* twinning. The talk began with a clear definition of the terms commonly used to define twins: *twinning by merohedry, twinning by reticular merohedry, twinning by pseudo-merohedry,* and *twinning by reticular pseudo-merohedry*. Some useful practical advice was offered on what to do before and during data collection in a twinned sample - including some simple but useful tips *e.g.* "Check that the crystal representative of the sample." The problem of overlapping and partially overlapping reflections was explained, and software for indexing these reflections was recommended. Victor presented three examples of *non-merohedral* twins, where the onset of twinning accompanied a phase transition, including the use of *Gemini, TwinUtil, TwinAbs* and *ROTAX*.

After a break for Lunch, Michael Ruf discussed the experimental aspects of handling twins, covering data collection, spot integration, indexing and absorption corrections). For data collection, advice included collection of lots of redundant data to aid the absorption correction and longer exposure times if one of the twin components is small. The latest version of *Saint* was previewed including lots of graphical feedback concerning the 3D spot shapes and overlap. The program *TwinAbs* can be used to apply separate absorption corrections to each twin domain and to merge equivalents, it will output HKLF4 (one component) data for structure solution and HKLF5 for twinned refinement in SHELXL. A further tool *RLatt* may be used to 'visually index' two or more domains by viewing and rotating the reciprocal lattice in real time and assigning reflections to different twin components. George Sheldrick's new *CELL_NOW* tool was also mentioned, this finds a potential cell and refines it using a robust least squares method, then searches for the same cell in the remaining non-indexed points using a rotation search.

Simon commenced the next hands-on session by demonstrating the handling of twinned data in CRYSTALS, including use of the ROTAX facility to detect twin laws missed during data collection, and stressed the importance of *coset decomposition* to detect the number of possible domains when dealing with twinning *via* a higher symmetry supercell. Delegates were provided with four twinned data sets to analyse in CRYSTALS, a *merohedral* twin, a *pseudo-merohedral* twin, and two *non-merohedral* twins (one of which had to be solved in the old-fashioned way from a Patterson map!) The use of ROTAX and the addition of a graphical interface make these steps very straight-forward within CRYSTALS to the extent that some of these could perhaps be described as 'routine twin analyses'.

While the hands-on session continued, Richard rounded off the afternoon with a demonstration of structure validation in CRYSTALS using the CCDC's Mogul software to generate statistics about the quality of a structure. Mogul is a library of geometric information (bond lengths, valence and torsion angles) for given chemical fragments, collected from the Cambridge Structural Database. These can be compared to a structure being refined to highlight incorrect or unusual features.

Fig 2.: *Workshop participants using crystals* (photos added by the newsletter editor)

CRYSTALS is free for academic and not-for-profit institutions from http://www.xtl.ox.ac.uk/crystals.html. As always, we gained a number of useful ideas from new users at the workshop about how the software can be improved, and we are grateful to everyone who participated. We would like to thank to Michael Ruf for an enlightening talk, David Watkin and Lachlan Cranswick for their help during the hands-on practical sessions, and Jeanette Krause Bauer for organising the workshop and provision of computers.

Richard Cooper
University of Oxford, Chemistry Research Laboratory, Mansfield Road, Oxford, OX1 3TA, UK. E-mail: richard.cooper@chemical-crystallography.oxford.ac.uk ; WWW: http://www.xtl.ox.ac.uk/

ECM-21, Durban, South Africa, 24th to 29th August 2003: Meeting Report: FA MS4 - Automatic Structure Determination: Challenges for the Future

Wouldn't it be nice: To possess a machine which is able to measure and solve crystal structures? You only have to press the “GO“-button and all crystal mounting, centering, measuring and structure solving and refining is done completely automatic. At least older crystallographers who have seen the developement from paper tape driven four circle diffractometers to ”smart“ CCD- or Imagine-Plate-machines, and from huge, punched card feeded computers in big computer centers to powerful workstations and PCs in our offices could imagine such a wonderful machine.

With some parts of these aspects dealt the microsymposium: “Automatic structure determination: Challenges for the future“ during the 21st European Crystallographic Meeting in Durban, South Africa. Not astonishing that this microsymposium started with the question: “We have taught computers crystallography – How do we teach them chemistry?” asked by **David Watkin** from Oxford University. At the begin of his talk David mentioned a citation given by J. S. Rollet that “methods have been developed by Ford, Hodgson, Rollet & Stonebridge for automatic solution of crystal structures“ more than 30 years ago. Some of this optimism has been justified by subsequent events, but David also said that at least every service crystallographer knows that fully automatic determination of crystal structures is still a dream. Such 'Black-Box' procedures can fail for several reasons: Bad crystals, poor diffraction data, disorder and twinning just to name some. David then stated that “crystallographic validators“ such as CHECKCIF can really only protect the crystallographer against inconsistencies in deposited data. The chemical correctness of a structure is connected with a proper answer to the following questions: Are the atom assignments correct? Is the stereo-chemistry correct? Is the connectivity correct? Especially the automatic atom assignment in inorganic crystal structures can sometimes be very difficult.

It should be mentioned that David started a collaboration between Oxford and the CCDC. This project links CRYSTALS with MOGUL and should lead one day to a new “chemical validator“. David summarized his talk with the memorable phrase: “When completed, Mogul will signal those structures which are either chemically very exciting, or perhaps just wrong“.

The second speaker was **Louis Farrugia** from the University of Glasgow on “The WinGX program system“. WinGX is a well known and widespread system of Windows programs suitable for the solution and refinement of single crystal diffraction data. It is primarily designed for small-molecule crystallography. It provides a consistent and user-friendly GUI for some of the best publicly available crystallographic programs, including the new SHELXD and SIR2002. This program system is also able to solve and refine crystal structures more or less completely automatic. Unfortunately, in the discussion of his talk Louis had to confess that WinGX has also some difficulties in working with incorrect molecular formula, especially with inorganic molecules.

“Automation in service and high throughput crystallography: Towards the 'dark laboratory'“ by **Simon Coles** from the University of Southampton was the first talk that also described the automated collection of data by means of robotic sample changing and implementation of software for diffractometer control and data processing without recourse to human intervention. The advances in this field of automation are really impressive. Even remote user interaction is possible by providing an interactive environment with staff and instrumentation through Web/Grid services. What bothers me, is that still someone else has to glue the single crystals on the top of the glass fiber with shellac or – in cases of low temperature measurements – with 'magic oil'.

The last speaker was **Ton Spek** from Utrecht University who also officiated as the session organizer of this microsymposium. Ton is author of the well known program PLATON that is a versatile multipurpose crystallographic tool. Historically PLATON started out as a program for the automated calculation of derived geometrical data for single crystal structures refined with SHELX76 at the end of the seventies. Over the time, various other tools like molecular graphics, all sorts of absorption corrections and data validation were added. Currently, the SYSTEM-S link in the program PLATON offers an attempt to automate a large part of the structure determination process along with easy access of a varity of software packages in the UNIX environment. SYSTEM-S can be regarded as a knowledgeable crystallographic shell around “state-of -the-art“ software packages including SHELXS, DIRDIF, SIR, SHELXL and the various facilities provided by PLATON.

The author of this meeting report uses the SYSTEM-S link for some years very successfully. Even with completely unknown compounds of good crystal quality from our Pharmaceutical Institute SYSTEM-S solves and refines the structures more or less automatic. However, the author prefers the *guided* mode instead of the so-called *no-questions-asked* mode. In this mode the user can choose one from all offered suggestions with the best suggestion set as default. That usually means the user has only to press the 'enter' key to proceed to the next step or cycle, but with obvious false decisions the user can immediately correct these decisions. Nonetheless, the correct assignment of atom types (e.g. O or N) is also often delicate, and the author can only finish the final structure refinement after the chemist has given the correct formula.

Although this microsymposium was the last official session at the ECM it was very well attended and very helpful

Jürgen Kopf
Institute of Inorganic and Applied Chemistry, University of Hamburg, Martin-Luther-King-Platz 6, D-20146 Hamburg, Germany. E-mail: kopf@xray.chemie.uni-hamburg.de - WWW: http://aclinux1.chemie.uni-hamburg.de/~xray/

ECM-21, Durban, South Africa, 24th to 29th August 2003: Meeting Report: Diffraction image processing and data quality

This symposium was held on Wednesday 26th August. The presentations addressed image processing and data quality in single crystal and powder diffraction, and covered both macromolecular and small molecule applications. The first presentation was given by Jim Pflugrath (Rigaku/MSC, USA), who focussed on the properties of modern area detectors and their relative advantages and disadvantages. CCD detectors are ubiquitous in small molecule applications, and are now becoming more common for macromolecular structure determinations. These detectors have a short readout time, in the range 1-10 s, with pixel sizes in the range 50-150 µm. However, they have a lower dynamic range than image plates, and are easy to saturate; they also show some energy dependence in their response. Image plates have been used, particularly by macromolecular crystallographers, for many years now. Their 'traditional' disadvantage, relative to CCD detectors, of a long readout time has recently been addressed, and may be as short as 10 s. They have a much higher dynamic range than CCDs, so that both strong and weak data may be reliably recorded on the same image. They also exhibit little energy dependence in their response, and they do not require cooling. Image plates are therefore becoming more competitive with CCDs, and modern developments include addition of some iodine to the phosphor to improve absorption, more powerful lasers to improve and expedite erasure and curved image plates.

Rob Hooft (Bruker Nonius BV, Delft, The Netherlands) described the EVAL series of programs (EVAL14, EVAL15 and EVALCCD) which encode a fundamental parameters approach to integration. Although reflections may be accurately located on a detector, determination of their shapes and boundaries is much more difficult. Even a spherical crystal may exhibit a range of spot shapes because of factors such as oblique incidence of the diffracted beam on the detector or α_1-α_2 splitting. In the fundamental parameters approach reflection profiles are derived directly from the shape of the source, the wavelength profile of the radiation, the crystal shape and its mosaicity. Since both the distribution of neighbouring reflections and the reflection profiles are well understood so too is the background. This applies even in difficult cases, such as twinned or incommensurate crystals. The wealth of background information available on an area detector means that elaborate models can be constructed, leading to improved estimates of standard uncertainties.

Harry Powell (MRC-LMB, Cambridge, UK) described the software written as part of the DNA project, which aims to automate data collection and processing. This is of particular importance at synchrotron beam-lines. The data collection part of the program is based on decisions that might be made by an experienced crystallographer after evaluation of crystal quality and determination of the unit cell. For example, the software automatically selects a crystal to detector distance, exposure time and scan range. Crystal decay and data quality are monitored during data collection. The program MOSFLM forms the data processing part of the DNA software. Once again, the aim is complete automation and the program can index an image, assess mosaic spread, refine the cell constants and integrate the data all without user intervention. SCALA is used for final scaling and merging. Initial tests show that DNA works well for 'normal' crystals; the modular design of the program will facilitate the incorporation of extra functionality.

Zbysek Dauter (Brookhaven National Laboratory, Upton, USA.) discussed how data quality may be assessed. Complete coverage of reciprocal space is extremely important and depends on the symmetry of the lattice. The data collection strategy needs to be considered very carefully. For example, though a 90° rotation is in principle enough to measure a complete data set from a crystal in point group 222, if the rotation is started from a position half way between the two-fold axes the data completeness may be as low as 60%. Some reflections may never fall into the diffracting position, though this may be minimised by off-setting a symmetry axis from the axis of rotation. Analysis of the quality of the intensity measurements and their associated standard uncertainties may be achieved with a range of statistical descriptors. The conventional R_{merge} is actually rather a poor measure because it tends to improve as redundancy increases. Better measures are those such as R_{meas}, R_{rim} and R_{pim}, which have been given by Diederichs & Karplus and Weiss & Hilgenfeld. These tended to be underused though, possibly because they adopt numerically higher values than the R_{merge} statistics that people are used to. Other measures have been designed specifically for anomalous data sets. Finally, though high redundancy was generally a Good Thing, merging together of data after significant crystal decay could degrade the quality of a data set.

Kenneth Shankland (RAL, Chilton, UK) spoke about data quality issues in powder diffraction, particularly as they relate to structure solution. Powders are studied not only because a material may only be available in this form, but also because a compound will usually be used in this form in industrial processes. Analysis is hampered by the overlap of symmetry inequivalent reflections with similar *d*-spacings, though collection of data at several temperatures can separate out overlapped peaks because by anisotropic thermal expansion. Much depends though on measuring data with good statistics. One way in which this can be achieved is to use a variable counting time for data collection, so that data at high angle may be measured for some 20 times as long as those at low angle. If good data are available, however, even relatively subtle effect such as the orientation of a sulfonamide group ($-SO_2NH_2$) can be determined. Recent innovations at the ESRF on beam-line ID31 (previously ID16) mean that data suitable for structure solution can be obtained in only a few minutes. However with the move from a bending magnet to an undulator, the beam intensity is now so great that radiation damage is a serious problem even in such short exposure times, and procedures for minimising this, for example summing of many quick scans, are currently under active investigation.

The organisers would like to thank the speakers for five excellent presentations.

Simon Parsons,
Crystallography Research Group, Department of Chemistry, University of Edinburgh, Joseph Black Building, The King's Buildings, West Mains Road, Edinburgh, EH9 3JJ, Scotland.
E-mail: s.parsons@ed.ac.uk ; WWW: http://www.chem.ed.ac.uk/staff/parsons.html

ECM-21, Durban, South Africa, 24th to 29th August 2003: Meeting Report: FA3 MS7: "Algorithms of the future. Chair: D Watkin, Co-chair: A Urzhumtsev

This microsymposium consisted of four lectures chosen to cover a wide range of interesting new developments in crystallography. The intension was to encourage the audience to look outside their normal range of interests, and perhaps show them things which might prove useful in their future work. The lecture room was almost filled to capacity, giving me the impression that people *are* interested in looking beyond their normal horizons.

1. Thanassis Hountas (Agricultural U. Athens, Greece) (gphy2xoa@aua.gr)
   "A solution of the phase problem by means of Quantum Mechanics. A novel approach or a next step?"

Comment:
I found this a mathematically very challenging lecture (ie I did not really understand it), but my overall impression was that this new technique offered an third 'space' in which phase refinement could be carried out. Refinements in the traditional spaces covered by the tangent formula and density modification are generally alternated since they have different reactions to errors. It remains to be seen if this third space can make a significant contribution to phase refinement. One apparently significant difference is that the probability formula for triplets does not contain the $1/N^{1/2}$ factor which applies to the tangent formula, so that the new formulation may be more robust for larger structures.

Abstract:

A solution of the phase problem by means of Quantum Mechanics. A novel approach or a next step? A. Hountas, K. Bethanis, P. Tzamalis[a] and G. Tsoucaris[b], [a] *Department of Sciences, Agricultural University of Athens, 75 Iera Odos Votanikos, Athens 11855, Greece,* [b] *Laboratoire de recherche des Musèes de France, Paris, France.*
Keywords: *Quantum Mechanical Crystallography.*

The underlying idea of the algorithm called "twin variables" [1] is the use of an auxiliary set of variables in addition to the classical set of normalized structure factors (SF) E's. This set $\Psi_\mathbf{H}$ consists of SF associated with a complex periodic function $\psi(\mathbf{r})$ such that $\rho(\mathbf{r}) = |\psi(\mathbf{r})|^2$. The two sets of the ***twin variables*** $\{E_\mathbf{H}, \Psi_\mathbf{H}\}$ are linked by:

(1) $E_\mathbf{H} = \Sigma_\mathbf{H}\Psi_\mathbf{K}(\Psi_\mathbf{K-H})^*$

(2) $\Psi_\mathbf{K} = \Sigma_\mathbf{H}E_\mathbf{H}\Psi_\mathbf{K-H}$ ($\Sigma_\mathbf{H} \equiv$ Sum over **H**).

The above equations can also be derived from first principles of Quantum Mechanics (QM) [2]. The QM formulation aims at the determination of the wave function for each electron based upon an extreme simplification of the QM potential function. We consider that there is no inter-electronic Coulomb repulsion, i.e. electrons are independent of each other. Thus, if $\mathbf{r_j}$ (j=1,..,N) are the positions of the N atoms in the unit cell and Z the atomic numbers, the potential and its Fourier Coefficients (FC) are:

$$V(\mathbf{r}) = -\Sigma_j (Z_j / |\mathbf{r}-\mathbf{r_j}|) \ \leftarrow FT \rightarrow \ W(\mathbf{H}) = -(1/\pi)\, \Sigma_j [Z_j \exp(2\pi i\, \mathbf{H}\cdot\mathbf{r_j}) / H^2]$$

Another key remark is that the FC of W(**H**) are simply related to the normalized SF $E_\mathbf{H}$ and the QM problem can be posed "determine the wave function $\psi(\mathbf{r})$ given a limited number of E's".

The electronic Schrödinger equation in momentum space provides by itself the following basis of a solution of the above problem:

$$(2\pi^2H^2-\varepsilon)\Psi_\mathbf{H} = (s_2/\pi)\Sigma_\mathbf{K} (E_\mathbf{H-K}\Psi_\mathbf{K} / |\mathbf{H-K}|^2)$$

Since the electron energy for bound states $\varepsilon<0$, this equation is very similar to (1) ($s_2 = (\Sigma_j Z_j^2)^{1/2}$).

Hydrogenoid atoms with the fundamental wave function of spherical symmetry and quasi-point electron density specify a satisfactory model for this purpose. Then, the SF $F_{\mathbf{H}}$ is given by:

$$F_{\mathbf{H}} = FT[|\psi(\mathbf{r})|^2] = c\Sigma_{\mathbf{K}}\Psi_{\mathbf{K}}(\Psi_{\mathbf{K-H}})^*$$

Where c is a scale factor and phase[$F_{\mathbf{H}}$] $\approx$ phase[$E_{\mathbf{H}}$].

Phase calculation in QM, and phase extension in “twin variables” procedures can be controlled by minimizing a constraint global function [1].

A very important problem is to establish reliable consistency criteria for the correctness of a phasing trial. In the present algorithm the introduction of a new criterion, based on the crystallographic symmetry, consists of testing the phase calculation, extension and refinement, by deliberately sacrificing the space group symmetry information in the auxiliary variable set, by using its gradual re-appearance as a criterion for correctness [3].

[1] Hountas A. and Tsoucaris G., *Acta Cryst.*, 1995, A**51**, 754-763.
[2] Bethanis K., Tzamalis P., Hountas A. and Tsoucaris G., *Acta Cryst.*, 2002, A**58**, 265-269.
[3] Tzamalis P., Bethanis K., Hountas A. and Tsoucaris G., *Acta Cryst.*, 2003, A**59**, 28-33.

2. Anders Markvardsen (RAL, Chilton, UK) (A.J.Markvardsen@rl.ac.uk)
" A Hydrid Monte Carlo algorithm for structure solution from powder diffraction data."

Comment:

The solution of the crystal structures of organic materials from the powder diffraction pattern and the empirical formula alone is still a long way off. However, if the structural formula is available, there is now a real chance of obtaining a convincing solution. The key to these new methods is in the highly optimized algorithms for comparing calculated and observed intensity data, and the methods for sampling conformational space.
Although primarily designed for the solution of structures from powder data, the methods can be effectively applied to single crystal data from poorly diffracting crystals which give an insufficient number of reflections above 1.2 A resolution for conventional direct methods to work.

Abstract:

A Hydrid Monte Carlo algorithm for structure solution from powder diffraction data.

A.J. Markvardsen, W.I.F. David and K. Shankland, *ISIS Facility, Rutherford Appleton Laboratory, Chilton, Oxon OX11 0QX, U.K.*

Keywords: *algorithm, hybrid Monte Carlo, powder diffraction*

The Hybrid Monte Carlo (HMC) algorithm first appeared in the field of quantum-chromodynamics [1] as a general purpose algorithm for the efficient sampling of a solution space. We have applied the algorithm to global optimization methods of structure solution from powder diffraction data [2]. In this context, the solution space is the collection of all possible structural configurations determined by the internal and external degrees of freedom of the molecular entities within a unit cell. Our experience with the HMC method indicates that it strongly outperforms more conventional algorithms such as simulated annealing.

Although I will demonstrate the HMC algorithm in the context of powder diffraction data, it is clearly of general applicability and its efficient sampling properties should be equally useful in other areas of crystallography that involve a global sampling or global minimization approach.

A detailed description of the key components of the HMC algorithm will be presented together with a comparison of this method with simulated annealing. Results will also be shown which demonstrate the use of the HMC method over a distributed computing network.

[1]Duane S, Kennedy A.D., Pendleton B.J., Roweth D., *Phys. Lett. B*, 1987, **195**, 216.

[2]Johnston J.C., David W.I.F., Markvardsen A.J., K. Shankland, *Acta Cryst.*, 2002, **A58**, 441.

3. Sinisa Prokic (ETH, Zurich, Switzerland) (sinisa@mat.ethz.ch)
"*Expol:* data analysis software for extracting more single-crystal-like intensities from data collected on a textured sample."

Comment:

Traditionally, preferred orientation of the crystallites in a powder sample has been seen as a hindrance to structure analysis. This very clever piece of work seeks to exploit variation in the 2-dimensional diffraction pattern recorded from orientated powder samples on a CCD detector as a function of sample orientation. By deliberately introducing preferred orientation into the sample, some of the degeneracy in the normal diffraction pattern can be reduced, leading to improved chances of structure solution. This is a technique which might have a significant future impact.

Abstract:

***Expol:* data analysis software for extracting more single-crystal-like intensities from data collected on a textured sample** Sinisa Prokic, Lynne B. McCusker and Christian Baerlocher,
Laboratory of Crystallography, ETH, CH-8092 Zurich, Switzerland
Keywords: *Structure solution, powder diffraction, texture, preferred orientation*

By exploiting the preferred orientation of the crystallites in a textured polycrystalline sample, the individual intensities of reflections that overlap in 2theta can be estimated more reliably and more single-crystal-like data can be obtained for structure solution.

Experiments performed on a textured sample in transmission mode using a 2D detector deliver a spatially extensive dataset. Typically 36 image plate frames, each corresponding to a 5° rotation of the sample (total of 180°), are measured. Each frame is integrated into 72 5° radial wedges by the program *Fit2d* [1], resulting in 2592 powder diffraction patterns corresponding to 1296 unique sample orientations.

In the first step of data analysis, an automatic intensity extraction procedure is used to obtain pole figure data (intensity change as a function of sample orientation) for non-overlapping reflections. The algorithm uses the SuperLU library [2] sparse matrix routines and can be controlled by convergence tests. The program must accommodate the refinement of 1296 patterns with complex varying backgrounds and geometrical distortions caused by the uncertainty in determining the center point of the scan and/or slight tilt of the imaging plate. To deal with the background problem, a pragmatic approach was chosen. A flexible pre-extraction procedure based on reliable background points in sparsely populated regions of the patterns was developed. The extraction results are saved after each stage, and poorly performing patterns are identified for manual inspection.

The pole figure data obtained from the extraction phase are then used to determine the orientation of the crystallites using a standard texture program. The ODF (Orientation Distribution Function) for the sample provides complete information about the spatial intensity variation of all reflections. This is then used in a procedure based on the Pawley algorithm [3] to refine a single set of reflection intensities (with esd's) that is consistent with all patterns. In the final step, the data are subjected to a Bayesian-statistics-based analysis [4] in order to treat any negative intensities in a statistically correct manner. The final result is a more-single-crystal-like set of intensities for structure solution.

[4] Hammersley, A. P., Svensson, S. O., Hanfland, M., Fitch, A. N., and Häusermann, D., "Two-Dimensional Detector Software: From Real Detector to Idealised Image or Two-Theta Scan", *High Pressure Research,* **14,** pp235-248 (1996)
[5] "A Supernodal Approach to Sparse Partial Pivoting", Demmel, J. W., Eisenstat, S. C., Gilbert, J. R., Li, X. S. and Liu, J. W. H., *SIAM J. Matrix Anal. Appl.,* vol. **20** (3), 720-755 (1999)
[6] Pawley, G. S., *J. Appl. Crystallogr.,* **14,** 357-61 (1981)
[7] Sivia, D. S. and David, W. I. F., *Acta Crystallogr. A*, **50,** 703-14. (1994)

4. Niels Volkman (The Burnham Institute, La Jolla, CA, USA) (niels@burnham.org)
"Docking of atomic models into reconstructions from electron microscopy."

Comment:
Electron microscopy is emerging as a powerful tool to complement X-ray diffraction and NMR studies of macromolecular materials. Recombination of focused electron diffraction data (images) of differently orientated samples yields the overall molecular envelope, though not at atomic resolution. However, if the sequence of the constituent subunits is known, it may be possible to fold the known structural formula to fit within the three dimensional envelope. This paper is concerned with the procedures needed to resolve alternative potential dockings of the proposed structure within the experimentally observed data

Abstract:

Reconstructions from electron microscopy. Niels Volkmann, Xiao-Ping Xu, Martin Fleming, Susanta Mukhopadhyay, and Dorit Hanein. *The Burnham Institute, 10901 North Torrey Pines Road, La Jolla, CA*
92037, USA. niels@burnham.org.

Keywords: *electron microscopy; docking; crossvalidation; statistical approach.*

In the three-dimensional structure determination of macromolecules, X-ray crystallography covers the full range from small molecules to very large assemblies with molecular masses of megadaltons. The limiting factors are expression, the stability and homogeneity of the structure, and subsequent crystallization. In the case of NMR, structures can be determined from molecules in solution, but the size limit, although increasing, is presently of the order of 100 kDa. Dynamic aspects can be quantified, but again the structures of mixed conformational states cannot be determined. Owing to dramatic improvements in experimental methods and computational techniques over the past few years, electron microscopy (EM) has matured into a powerful and diverse collection of methods that allow the visualization of the structure and dynamics of an extraordinary range of macromolecular assemblies at resolution spanning from molecular to near atomic. In addition, cryo-methods enable the observation of molecules close to physiological conditions in their native aqueous environment. While not hampered by many of the limitations of NMR or crystallography, EM imaging is limited to lower resolution for most biological specimens (10-30Å), thus precluding atomic modeling directly from the data. Still, atomic models often can be generated by combining high-resolution structures of individual components in a macromolecular complex with a low resolution structure of the entire assembly. Analysis of the resulting models can lead to new hypotheses and contribute important insights into interaction and regulation of the individual molecular components. Thus the combination of atomic-resolution structures with EM provides a powerful tool to gain insight into cellular processes. We will present an integrated, automatic approach for such combination and the associated analysis. The approach is primarily based on a statistical evaluation of all possible docking solutions consistent with the underlying data and avoids overfitting artifacts by intensive use of cross-validation.

David Watkin
Chemical Crystallography, The Basement, Department of Chemistry, University of Oxford, Chemistry Research Laboratory, Mansfield Road, Oxford, OX1 3TA, England
E-mail: david.watkin@chemistry.oxford.ac.uk ; WWW: http://www.xtl.ox.ac.uk/

Call for Contributions to the Next CompComm Newsletter

The forth issue of the Compcomm Newsletter is expected to appear around July of 2004 with the primary theme still to be determined. If no-one is else is co-opted, it will be edited by Lachlan Cranswick.

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

Please send articles and suggestions directly to the editor.

Lachlan M. D. Cranswick
NPMR, NRC,
Building 459, Station 18,
Chalk River Laboratories,
Chalk River, Ontario,
Canada, K0J 1J0
E-mail: lachlan.cranswick@nrc.gc.ca