Reciprocal Space Tutorial

IUCr Computing School, Siena,
August 2005

George M. Sheldrick

http://shelx.uni-ac.gwdg.de/SHELX/

Fortran-90

Despite the massive campaigns for python etc., we should not forget that Fortran was specifically designed for scientific number crunching applications, and – especially Fortran-90 – still has many advantages for crystallographic calculations:

1. Fortran code is extremely portable and stable – the original SHELX-76 code still compiles and runs correctly on any modern computer without any changes being necessary.
2. It is easy to produce robust stand-alone statically-linked binaries with ZERO DEPENDENCIES (e.g. SHELX).
3. Many highly optimized and debugged libraries are available for numerical applications (e.g. the Intel MKL, which includes multithreaded multidimensional mixed-radix FFTs).
4. Fortran-90 added many useful new features, e.g. array and character operations, runtime memory allocation etc.
5. OPEN-MP enables multithreaded code to be generated easily.

Fortran-90 example

The following Fortran-90 program is provided as a solution to Kevin's first exercise [finding separate domains in a cyclic mask consisting of 0 (solvent) and 1 (protein)]. It makes limited use [e.g. read(*,*)m; m=-m; p=minloc(m)] of F-90 array functions:

```
integer::m(12,10,8),p(3),c(960)
read(*,*)m; m=-m; n=0
1 p=minloc(m); if(m(p(1),p(2),p(3)).ge.0)goto 7
n=n+1; c(n)=1; m(p(1),p(2),p(3))=n
2 l=0; do 5 k=1,8; do 4 j=1,10; do 3 i=1,12
if(m(i,j,k).ge.0)goto 3
if(max(m(mod(i,12)+1,j,k),m(mod(i+10,12)+1,j,k),&
m(i,mod(j,10)+1,k),m(i,mod(j+8,10)+1,k),&
m(i,j,mod(k,8)+1),m(i,j,mod(k+6,8)+1)).le.0)goto 3
m(i,j,k)=n; c(n)=c(n)+1; l=1
3 continue; 4 continue; 5 continue
if(l.gt.0)goto 2; goto 1
6 format(' Domain',i4,'  contains',i5,'  pixels')
7 write(*,6)(i,c(i),i=1,n); end
```

What is an algorithm ?

An algorithm is a way of calculating something that even the person who invented it cannot understand !?

Usually algorithms involve some clever mathematics to do something faster – sometimes by orders of magnitude – or 'better' than the obvious way of doing it.

Examples:

Sorting (e.g. reflection and peak lists)

FFT for structure factor calculations etc.

Fast calculation of interatomic distances

Derivation of phase relations for direct methods

Simple example – mean and variance

$$\bar{x} = \sum x / N \qquad V = \sum (x - \bar{x})^2 / N$$

This appears to require two scans through the data: the first to find $\bar{x}$ and the second to find **V**. A 'one-scan' method would be faster if the data are stored on a disk (or are in RAM but do not all fit into the CPU 'cache' at the same time). We will look into this trivial example in detail because it illustrates several important points.

A one-scan algorithm

$$V = \sum (x - \bar{x})^2 / N = \sum (x^2 - 2x\bar{x} + \bar{x}^2) / N$$

But $\bar{x} = \sum x / N$. Substituting this:

$$V = \sum x^2/N - [(\sum x)/N]^2$$

Which only requires one scan, summing both $\sum x$ and $\sum x^2$.

Usually we need $\sqrt{V}$, so it is important to know whether it is ever possible for **V** calculated in the above way to be negative, this would 'crash' the program! In a mathematical sense **V** can never be negative, but it could still happen as a result of rounding errors if all **x** are equal and non-integral. So a good 'defensive' programmer would intuitively avoid this by replacing a negative value of $\sum x^2/N - [(\sum x)/N]^2$ by zero.

The hidden trap

If the floating point numbers are (as is often the case) being stored in 4 bytes each, they will have a precision of about 6½ decimal digits. If we happen to be summing over 10^8 pixels of a large Fourier map to calculate the 'one sigma level' $\sqrt{V}$, both formulas will give the wrong answers !!

The reason is that when we add (say) the 10000000th pixel to the running totals, the increment will be smaller than the precision to which the numbers are being stored, so the running totals will not change, i.e. the increment is thrown away.

Possible remedies are to sum rows and layers of the map first and then add the totals, or do all the calculations with 8-byte (64-bit) numbers.

Sort algorithms

Sorting algorithms are the highlight of many informatics courses because the difference in speed can be enormous. An obvious method such as scanning A(1..N) to find the largest element, then swapping it with A(1), then scanning A(2..N) and swapping the largest element with A(2), then scanning A(3..N) etc. takes a time of order N^2 (which for large N can be all week). The best general algorithms are of order NlogN but are complicated (best to call a library routine).

Sometimes – as with sorting reflection lists – we can take advantage of the special features of the particular problem to reduce the order to N. For 10000 reflections, N^2 is 100000000, NlogN is 50000 but N is only 10000 (but the constant factor multiplying the order may differ)!

A typical sort algorithm

The following FORTRAN routine (called comb-sort) is a good general purpose algorithm for sorting a few hundred items (e.g. Fourier map peaks). Although not quite NlogN it is fast because it has a low overhead. The array A(1..N) (containing e.g. peak heights) is sorted into descending order. In practice the IF..ENDIF loop would also need to swap the atom coordinates x, y and z as well as the peak heights.

```
      K=N
1     K=INT(REAL(K)/1.2796)
      IF(K.LT.1)K=1
      IF(K.EQ.9.OR.K.EQ.10)K=11
      M=0
        DO 2 I=1,N-K
        J=I+K
        IF(A(J).GT.A(I))THEN
          Q=A(J)
          A(J)=A(I)
          A(I)=Q
          M=1
          ENDIF
2       CONTINUE
      IF(M+K.GT.1)GOTO 1
```

Sorting reflection lists (order N)

First h,k,l are transformed to a standard equivalent (e.g. maximum l, if l values are equal then maximum k, if both k and l equal then maximum h). Then the maximum and minimum values of each index are found.

To sort on h, scan list, count how often each h is present, storing the results in an integer array $N(h_{min}..h_{max})$. This is then converted so that it holds 'pointers' p_h to the final list:

	↓p_3	↓p_4	↓p_5	↓p_6	FINAL LIST
• • •	all h=3	all h=4	all h=5	all h=6	• • •

The list is scanned again, putting each reflection into the final location pointed to by p_h and then incrementing p_h.

The list is sorted first on h, then on k, and finally on l. In the final sorted list, equivalents finish next to each other and so can easily be averaged.

Sorting small integers (Fortran-77)

```
C
      SUBROUTINE INSORT(N,IP,IQ,ID)
C
C Sort-merge integer data in order of ascending ID(I). IP is the
C current pointer array to ID and IQ becomes the new pointer array.
C
      INTEGER::IP(N),IQ(N),ID(N),IT(999)
      L=ID(1)
      M=L
        DO 1 I=2,N
        L=MIN0(ID(I),L)
        M=MAX0(ID(I),M)
1     CONTINUE
      L=L-1
      M=M-L
        DO 2 I=1,M
        IT(I)=0
2     CONTINUE
        DO 3 I=1,N
        J=ID(I)-L
        IT(J)=IT(J)+1
3     CONTINUE
      J=0
        DO 4 I=1,M
        K=J
        J=J+IT(I)
        IT(I)=K
4     CONTINUE
        DO 5 I=1,N
        J=ID(IP(I))-L
        IT(J)=IT(J)+1
        IQ(IT(J))=IP(I)
5     CONTINUE
      RETURN
      END
```

Sorting and merging reflections

Kevin Cowtan's notes on Symmetry in Reciprocal Space must be read before attempting this exercise!

The file "in" contains cell, symops and reflection data in free format for a small protein in space group $P3_121$. There are 389596 data before merging. The exercise is to sort-merge the data, remove systematic absences (and print them out with I/σ ratios) and count the number of unique reflections remaining, and how many of them are in centric projections (and so have no anomalous differences). R(int) = $\sum |I - \langle I \rangle| / \sum I$ should also be calculated (only reflections in groups of more than one equivalent should be included in the R(int) calculation).

There should be 42827 remaining unique reflections, 4354 of them centric and R(int) is 0.0466.

Any computer language and libraries may be used.

The symmetry operators

All symmetry-dependent information in real or reciprocal space can be derived from the symmetry operators! E.g. Space group $P3_1$:

m = 1: x, y, z; m=2: -y, x-y, z+1/3; m=3: -x+y, -x, z+2/3

These operators may also be expressed as 3x3 matrices R plus vectors t:

$$\begin{pmatrix} x_m \\ y_m \\ z_m \end{pmatrix} = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}$$

Or: $x_m = R_m x + t_m$, which in the case of operator m=3 is:

$$\begin{pmatrix} x_m \\ y_m \\ z_m \end{pmatrix} = \begin{pmatrix} -1 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 2/3 \end{pmatrix}$$

Symmetry operators for $P4_12_12$

For the examples in this talk we will use the space groups $P3_1$ and $P4_12_12$; for the latter the general positions are:

m=1: x, y, z m=2: –x, –y, z+½

m=3: ½–y, ½+x, z+¼ m=4: –y, –x, ½-z

m=5: ½+y, ½–x, z+¾ m=6: ½–x, ½+y, ¼–z

m=7: ½+x, ½–y, ¾–z m=8: y, x, -z

i.e. for m=5:

$$R = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad t = \begin{pmatrix} ½ \\ ½ \\ ¾ \end{pmatrix}$$

To obtain the general positions of the enantiomorphous space group $P4_32_12$, just exchange ¼ and ¾ !

Properties of R and t

The ***determinant*** of the matrix R must be +1 or –1. If it is –1 it produces an inverted image, so the space group in ***not chiral*** (but may still be non-centrosymmetric).

When one row of R is never negative for any operator [e.g. the third row (R_{31} R_{32} R_{33}) in $P3_1$] the space group is ***polar***.

If all elements of t are zero for all operators (not including lattice centering) the space group is ***symmorphic*** (and has no systematic absences apart from lattice absences).

If t includes elements that are **not multiples of ½** and the lattice is **primitive** the space group is ***one member of an enantiomorphic pair***; if either all elements of t are multiples of ½ or the lattice is centered the space group does not belong to an enantiomorphic pair (!)

Symmetry in reciprocal space

For the symmetry operator m: $x_m = R_m x + t_m$

The calculated structure factor F_c is given by the complex number $F_c = (A + iB)$ where:

$$A_{hkl} = \sum_{atoms} \sum_{symm} f_j \cos[2\pi(hx_m+ky_m+lz_m)]$$

$$B_{hkl} = \sum_{atoms} \sum_{symm} f_j \sin[2\pi(hx_m+ky_m+lz_m)]$$

(the exponential term for atomic displacements has been included in the scattering factor f_j here for simplicity). But

$$hx_m+ky_m+lz_m = h(R_m x+t_m) = h_m x + k_m y + l_m z + ht_1 + kt_2 + lt_3$$

where: $h_m = R_{11}h + R_{21}k + R_{31}l$
$k_m = R_{12}h + R_{22}k + R_{32}l$
$l_m = R_{13}h + R_{23}k + R_{33}l$

So to find the ***equivalent indices*** h_m, k_m, l_m we multiply h, k, l by the ***transpose*** of the matrix R.

The phases of equivalent reflections

The phase ϕ_m of the equivalent reflection h_m is derived from the phase ϕ of the (prime) reflection h by:

$$\phi_m = \phi - 2\pi h t_m = \phi - 2\pi (ht_1 + kt_2 + lt_3)$$

For example in $P3_1$: $h_2 = 0.h + 1.k + 0.l = k$
$k_2 = -1.h - 1.k + 0.l = -h - k$
$l_2 = 0.h + 0.k + 1.l = l$

So h_2 is k, –h–k, l with phase:

$$\phi_2 = \phi - 2\pi h t_2 = \phi - 2\pi (h.0 + k.0 + l.1/3) = \phi - 2\pi l/3$$

These are the true equivalent reflections; they have the same intensities and exactly the above phase shifts whether anomalous scatterers (that would cause Friedel's law to break down) are present or not.

Friedel's law

Friedels law states that $|F_{-m}| = |F_m|$ and $\phi_{-m} = -\phi_m$ where ϕ_{-m} is the phase of $-h_m -k_m -l_m$. Friedel's law is strictly valid only when f'' is equal (or zero) for all atoms in the structure, but it is almost always a good approximation. In space group $P3_1$:

$|F_{h,k,l}| = |F_{k,-h-k,l}| = |F_{-h-k,h,l}|$ (exact equivalents) and

$|F_{-h,-k,-l}| = |F_{-k,h+k,-l}| = |F_{h+k,-h,-l}|$ (exact equivalents)

but these two groups are only approximately equal because they are related by Friedel's law. For non-centrosymmetric space groups (chiral or not) there are always two groups of exact equivalents; if Friedel's law holds, the |F| values of the two groups are also the same.

Equivalents in $P4_12_12$

For $P4_12_12$ the two groups of equivalents are:

h,k,l = –h,–k,l = k,–h,l = –k,–h,–l = –k,h,l = –h,k,–l = h,–k,–l = k,h,–l

–h,–k,–l = h,k,–l = –k,h,–l = k,h,l = k,–h,–l = h,–k,l = –h,k,l = –k,–h,l

The space group P4mm has the same Laue group as $P4_12_12$ but different Friedel-related groups:

h,k,l = –h,–k,l = k,–h,l = –k,–h,l = –k,h,l = –h,k,l = h,–k,l = k,h,l

–h,–k,–l = h,k,–l = –k,h,–l = k,h,–l = k,–h,–l = h,–k,–l = –h,k,–l = –k,–h,–l

To derive these groups of equivalents correctly, it is necessary to know the point group (or space group). The Laue group contains an inversion center and so is not sufficient. It should be noted that for chiral compounds (i.e. for macromolecules) there is only one possible point group (the one that has rotation axes but no mirror planes, inversion centers or inverse tetrads) corresponding to each Laue group.

Systematic absences

A reflection is ***systematically absent*** when $h_m = h$ but ϕ_m is not equal to ϕ ($+2n\pi$ where n is an integer). In $P3_1$:

$$\phi_{k,-h-k,l} = \phi_{h,k,l} - 2\pi l/3 \ (+2n\pi)$$

so when h = k = 0:

$$\phi_{0,0,l} = \phi_{0,0,l} - 2\pi l/3 \ (+2n\pi)$$

which can only be true when l = 3n, i.e. reflections 0,0,l with l not equal to 3n are ***systematically absent***.

Note that the reflection is absent if this applies for ***any*** operator m. E.g. in $P4_12_12$:

$$m=2: \quad \phi_{-h,-k,l} = \phi_{h,k,l} - \pi l \ (+2n\pi)$$

which implies 0,0,l absent for l not equal to 2n, but:

$$m=3: \quad \phi_{k,-h,l} = \phi_{h,k,l} - \pi h - \pi k - \tfrac{1}{2}\pi l \ (+2n\pi)$$

which requires 0,0,l absent for l not 4n, so 0,0,2 is also absent.

Symmetry-restricted phases

If $h_{-m} = h$ but ϕ_{-m} is not equal to ϕ ($+2n\pi$) then:

$$\phi_{-m} = -(\phi - 2\pi h t_m) = \phi \ (+2n\pi)$$

which gives the equation:

$$2\phi = 2\pi h t_m + 2n\pi \ \text{(n integer)}$$

$$\text{or} \quad \phi = \pi(ht_1 + kt_2 + lt_3 + n)$$

So there can only be two possible values for ϕ (corresponding to n odd and n even) and they must differ by π. Such reflections belong to a ***centrosymmetric projection***. In $P3_1$, for no values of m and h,k,l (except 0,0,0) is $h_{-m} = h$, so there are no centrosymmetric projections. This is clearly also true in real space from inspection of the IT diagram.

Centric reflections in $P4_12_12$

$P4_12_12$ has several classes of reflections with restricted phases, e.g.:

$$m=2: \quad h_{-m} = -(-h,-k,l)$$

which is equal to h,k,l when l=0, which gives:

$$\phi_{h,k,0} = \pi(h.0 + k.0 + 0.\tfrac{1}{2}) + n\pi = n\pi$$

so the h,k,0 reflections have phases restricted to 0 or π.

Similarly, m=6 gives $h_{-m} = -(-h, k, -l)$ which is equal to h,k,l if k = 0. Then:

$$\phi_{h,0,l} = \pi(h.\tfrac{1}{2} + 0.\tfrac{1}{2} + l.\tfrac{1}{4}) + n\pi = \pi(\tfrac{1}{2}h + \tfrac{1}{4}l) + n\pi$$

So for example the reflection 1,0,1 has two possible phases of $3\pi/4$ or $7\pi/4$.

In the case m=4, $h_{-m} = -(-k,-h,-l)$ which is equal to h,k,l when h = k. Thus it can be shown that reflections h,h,l are restricted to $\pi/2$ or $3\pi/2$.

Datafile for sort-merge exercise

The file "in" begins with a comment line, followed by the cell, symmetry and reflection data all in free format.

```
Trigonal bovine trypsin P3121 #152 CuKa
54.735 54.735 106.786 90 90 120
6 symops follow, then h,k,l,I and sig(I)
1 0 0 0 1 0 0 0 1 0.0 0.0 0.0
0 -1 0 1 -1 0 0 0 1 0.0 0.0 0.333333
-1 1 0 -1 0 0 0 0 1 0.0 0.0 0.666667
0 1 0 1 0 0 0 0 -1 0.0 0.0 0.0
1 -1 0 0 -1 0 0 0 -1 0.0 0.0 0.666667
-1 0 0 -1 1 0 0 0 -1 0.0 0.0 0.333333
22 -3 -41 21.38 3.27
-2 6 -12 162.92 11.71
-19 4 -32 81.44 6.82
-13 -9 -51 16.44 3.87
```

etc. 389596 reflections in total, terminated by the end of the file.

Hints for the tutorial

1. Remember to use ***transposed*** symops for transforming the reflections to their equivalents! Do not forget Friedel's law.
2. The basic idea is to tranform all reflections to a standard setting (e.g. maximum *I*, if *I* is equal for two equivalents then maximum *k* etc.). This list of standardized reflections is then sorted so that reflections with the same indices appear adjacent in the sorted list. It is quicker to sort pointers than shuffling the contents of the lists.
3. Reflections with same indices are merged: $\langle I \rangle = \sum I / n$; $\sigma(\langle I \rangle) = 1 / [\sum(1/\sigma^2(I))]^{\frac{1}{2}}$ [for more sophisticated merging see Blessing, *J. Appl. Cryst.* 30 (1997) 421-426].
4. To find out if a reflection is systematically absent, generate equivalent *h,k,l* and see if any are identical to the original. A non-integral phase shift (calculated using *h,k,l* and the translational part of the symop.) then indicates an absence. Centric reflections have equivalents with indices *–h,–k,–l*.

Fortran-77 and C++ solutions to exercise

Solutions to the exercise are provided in both Fortran-77 (*smerg.f*) and C++ (*sortmerge.cpp*, written by Tim Grüne). Note that the C++ file-reading routine could be replaced by the appreciably faster C-routine in the latter. Under Linux they may be compiled and run as follows:

```
f77 smerge.f –o smerg
time ./smerg <in
```

```
g++ sortmerge.cpp –o sortmerge
time ./sortmerge in
```

which gave runtimes of 2.3 and 2.8 seconds resp. on a 3 GHx Xeon. After optimization (in both cases with –O –ffast-math) these were reduced to 1.9 and 2.3 seconds. The Intel ifort compiler was (in this somewhat untypical example) a little slower.