TOPAS, TOPAS-Academic

Technical Reference

Version 7

Alan A Coelho *www.topas-academic.net*

Bruker–AXS *www.bruker-axs.de*

September 7, 2020

Ab initio solution of proteins at atomic resolution, Fast simultaneous refinement of 1000s of data sets, Amazon EC2 cloud computing, PDF Generation, Deconvolution, Capillary aberration, LP-Search, Sine Transform, DPI awarenessPeak fitting, Pawley & Le Bail refinement, Rietveld refinement, PDF Generation, PDF refinement, Magnetic structures, CW Neutron refinement, TOF refinement, Stacking-faults, Laue refinement, Indexing, Charge flipping, Structure solution, Deconvolution and $K\alpha_1$ stripping, Penalties, Restraints

Contents

1. ..INTRODUCTION

This document describes the kernel operation of TOPAS-Academic including its macro language. The kernel is written in ANSI c++ with internal data structures comprising a tree similar-to an XML representation. Individual tree nodes correspond to c++ objects; understanding the internal structures facilitates program operation. Input is through an input file (*.INP) comprising readable keywords and macros, the latter being groupings of keywords. The kernel pre-processes the INP file expanding macros as required; the resulting pre-processed file (written to TOPAS.LOG) comprises keywords that are operated on by the kernel. On parsing the INP file the kernel creates its internal data structures. The main tree-node objects are:

> *xdd...*
> > *bkg*   : Background.
> > *str...*   : Structure information for Rietveld refinement.
> > *xo_ls...*   : 2θ-I values for single line or whole powder pattern fitting.
> > *d_ls...*   : d-I values for single line or whole powder pattern fitting.
> > *hkl_ls...*   : Lattice information for Le Bail or Pawley fitting.
> > *fit_obj...* : User defined fit models.

str, *xo_ls*, *d_ls* and *hkl_ls* are referred to as "phases" and the peaks of these "phase peaks". A listing of the data structures is given in section 18.1.

1.1..... Conventions

- *Keywords look like this*.
- *Macros look like this*.
- Keywords enclosed in square brackets [ ] are optional.
- Keywords ending in ... indicate that multiple keywords of that type are allowed.
- Text beginning with the character # corresponds to a number.
- Text beginning with the character $ corresponds to a User defined string.
- E after keyword: An equation (i.e. = a+b;) or constant (i.e. 1.245) or a parameter name with a value (i.e. lp 5.4013) that can be refined.
- !E after keyword: An equation or constant or a parameter name with a value that cannot be refined.

To avoid input errors, it is useful to differentiate between keywords, macros, parameter names, and reserved parameter names. The conventions followed are:

Keywords :	all lower case
Parameter names :	first letter in lower case
Macro names :	first letter in upper case
Reserved parameter names :	first letter in upper case

1.2..... Input file example (INP format)

The following is an example input file for Rietveld refinement of corundum and fluorite:

```
' Rietveld refinement comprising two phases
xdd File_Name.xy
    CuKa5(0.001)                        ' Emission profile
    Radius(217.5)                       ' Diffractometer radius
    LP_Factor(26.4)                     ' Lorentz polarization
    Slit_Width(0.1)                     ' Receiving slit width
    Divergence(1)                       ' Equatorial divergence
    Full_Axial_Model(12, 15, 12, 2.3, 2.3) ' Axial divergence
    Zero_Error(@, 0)
    bkg @ 0 0 0 0 0 0
    STR(R-3C, "Corundum Al2O3")
        Trigonal(@ 4.759, @ 12.992)
        site Al x 0       y 0  z @ 0.3521  occ Al+3 1  beq @ 0.3
        site O  x @ 0.3062 y 0  z   0.25    occ O-2  1  beq @ 0.3
        scale @ 0.001
        CS_L(@, 100)
        r_bragg 0
    STR(Fm-3m, Fluorite)
        Cubic(@ 5.464)
        site Ca    x 0      y 0     z 0       occ Ca 1  beq @ 0.5
        site F     x 0.25  y 0.25 z 0.25    occ F  1  beq @ 0.5
        scale @ 0.001
        CS_L(@, 100)
        r_bragg 0
```

The format is case sensitive. Optional indentation can be used to show tree dependencies. Placement of keywords within a tree level is not important. For example, the keyword *str* signifies that all information (pertaining to *str*) occurring between this keyword and the next level of the same type (in this case *str*) applies to the first *str*. All input text streams can have line and/or block comments. A line comment is indicated by the character ' and a block comment by an opening /* and closing */. Text from the line comment character to the end of the line is ignored. Text within block comments are ignored; block comments can be nested. Here are some examples:

```
' This is a line comment
space_group C2/c ' This is also a line comment
/* This is a block comment.
    A block comment can comprise any number of lines. */
```

On termination of refinement an output file (*.OUT) similar-to the input file is created with refined values updated.

1.3..... Test examples

The directory TEST_EXAMPLES contain examples that can act as templates for creating INP files. In addition, charge-flipping examples are found in the CF directory and indexing examples in the INDEXING directory.

1.4..... TC-INPS.BAT and the aac$ macro

The batch file TC-INPS.BAT runs through over 150 test examples in a few minutes. These examples play an important role in program testing. Arguments passed via the command line to the test

examples can contain the *aac$* macro; if defined *aac$* is expanded at the bottom of the INP file. For example, to terminate refinement after 100 iterations the following could be used:

    tc test_examples\pdf\alvo4\rigid "macro aac$ { iters 100 verbose 0 }"

1.5 ..... TOPAS is 64 bit

The command line TC.EXE and the GUI TA.EXE both run on the Windows 64-bit operating system. This means all available memory can be used.

1.6 ..... Limiting Memory Usage – MaxMem.TXT

Accidental INP file errors coupled with 64-bit address space can lead to excessive memory usage. A wrong decimal place in a lattice parameter for example could lead to the generation of billions of hkls. In cases where all of RAM is used, the Windows 7 and Windows 10 operating systems hang with the task-manager becoming unresponsive. The reason for the 'hang' is due to the swapping of data/programs to and from virtual memory (typically a swap file on the hard disc) by the operating system. This 'hang' scenario is typically avoided using option (1) below which is the default. The file MAXMEM.TXT, found in the main TOPAS directory, comprises two floating point numbers A and B and their use is as follows (all values in Gbytes):

1)   If A=0, the maximum allowed memory usage is set to 75% of physical memory.

2)   If the number in MAXMEM.TXT is negative, the maximum allowed memory usage is:

    Max_Mem_Allowed = Max_Physical_Memory + A

3)   If the number in MAXMEM.TXT is positive, the maximum allowed memory usage is:

    Max_Mem_Allowed = A

The default value in MAXMEM.TXT is zero which corresponds to case (1). For all cases, if memory usage exceeds Max_Mem_Allowed then TC.EXE aborts with the message:

    Memory allocation limit reached, increase limit in file MaxMem.TXT

TA.EXE also aborts with a message and additionally it creates the empty file called MAXMEM-CHK.TXT. Checking the time/date stamp of MAXMEM-CHK.TXT reveals whether TA.EXE aborted due to excessive memory usage.

2. ..PARAMETERS

2.1..... When is a parameter refined

A parameter is flagged for refinement by giving it a name. The first character can be an upper or lower-case letter. Subsequent characters can include the underscore character '_' and the numbers 0 through 9. For example:

```
site Zr x 0 y 0 z 0 occ Zr+4 1 beq b1 0.5
```

Here b1 is a name given to the *beq* parameter. No restrictions are placed on the length of parameter names. The character ! placed before b1, as in !b1, signals that b1 is not to be refined, for example:

```
site Zr x 0 y 0 z 0 occ Zr+4 1 beq !b1 0.5
```

A parameter can also be flagged for refinement by placing the @ character at the start of its name. Internally the parameter is given a unique name and treated as an independent parameter. The b1 text in the following is ignored:

```
    site Zr x 0 y 0 z 0 occ Zr+4 1 beq @ 0.5
or, site Zr x 0 y 0 z 0 occ Zr+4 1 beq @b1 0.5
```

2.2..... User defined parameters - the prm keyword

The [*prm*|*local* E] keyword defines a new parameter. For example:

```
prm b1 0.2 ' b1 is the name given to this parameter
           ' 0.2 is the initial value
site Zr x 0 y 0 z 0 occ Zr+4 0.5  beq = 0.5 + b1;
                    occ Ti+4 0.5  beq = 0.3 + b1;
```

Here b1 is a new parameter that will be refined; this example demonstrates adding a constant to a set of *beq*'s. Note the use of the '=' sign after the *beq* keyword; this indicates that the parameter is in the form of an equation. In the following example, b1 is used but not refined:

```
prm !b1 0.2
site Zr x 0 y 0 z 0 occ Zr+4 0.5  beq = 0.5 + b1;
                    occ Ti+4 0.5  beq = 0.3 + b1;
```

Parameters can be assigned the following attribute equations that can be functions of other parameters:

[*min* !E] [*max* !E] [*del* !E] [*update* !E] [*stop_when* !E] [*val_on_continue* !E]

min and *max* can be used to limit parameter values during refinement, for example:

```
prm b1 0.2 min 0 max = 10;
```

Here b1 is constrained to within the range 0 to 10. *min* and *max* can be equations themselves and can therefore be functions of other parameters. Limits are very effective in refinement stabilization. *del* is used for calculating numerical derivatives with respect to the calculated pattern; typically, internal default *del* values are adequate. Parameter values are updated at the end of an iteration as follows:

new_Val = old_Val + *Change*

When *update* is defined then the following is used:

new_Val = "update equation"

update can additionally be a function of the reserved parameter names *Change* and *Val*. The use of *update* does not negate *min* and *max*. *stop_when* is a conditional statement used as a stopping criterion. In this case convergence is determined when *stop_when* evaluates to a non-zero value for all defined *stop_when* attributes, as defined for independent parameters, and when the *chi2_convergence_criteria* condition has been met. *val_on_continue* is evaluated when *continue_after_convergence* is defined. It provides a means of changing parameter values after refinement convergence where:

new_Val = *val_on_continue*

Here are example attribute equations as applied to the *x* parameter:

```
x @ 0.1234
    min       = Val - 0.2;
    max       = Val + 0.2;
    update    = Val + Rand(0, 1) Change;
    stop_when = Abs(Change) < 0.000001;
```

2.3..... Parameter constraints

Equations can be a function of parameter names; this provides a mechanism for introducing linear and non-linear constraints, for example:

```
site Zr x 0 y 0 z 0 occ Zr+4 zr 1     beq 0.5
                    occ Ti+4 = 1-zr; beq 0.3
```

Here the zr parameter is used in the equation "= 1-zr;"; this equation defines the Ti+4 site occupancy. Note, equations start with an equal sign and end in a semicolon. Limiting zr with *min*/*max* can be performed as follows:

```
site Zr x 0 y 0 z 0 occ Zr+4 zr 1 min 0 max 1  beq 0.5
                    occ Ti+4 = 1-zr;           beq 0.3
```

Here zr will be constrained to within 0 and 1. An example for constraining the lattice parameters *a*, *b*, *c* to the same value as required for a cubic lattice is as follows:

```
a lp 5.4031   b lp 5.4031   c lp 5.4031
```

Parameters with names that are the same must have the same value. An exception is thrown if the lp parameters were defined with different values. Another means of constraining the three lattice parameters to the same value is by using equations with the parameter lp defined only once, or,

```
a lp 5.4031   b = lp;      c = lp;
```

More general again is the use of the Get function as used in the *Cubic* macro:

```
a @ 5.4031    b = Get(a);  c = Get(a);
```

Here the constraints are formulated without the need for a parameter name.

2.4..... The *local* keyword

The *local* keyword is used for defining named parameters as local to the top, xdd or phase level; *local* can simplify complex INP files. The following code fragment:

```
xdd    local a 1
xdd    local a 2
```

has two 'a' parameters; one dependent on the first *xdd* and the other dependent on the second *xdd*. Internally two independent parameters are generated, one for each of the 'a' parameters; this is necessary as the parameters require separate positions in the **A** matrix for minimization, correlation matrix, errors etc... In the code fragment:

```
local a 1                 ' top level
xdd   gauss_fwhm = a;     ' 1st xdd
xdd   gauss_fwhm = a;     ' 2nd xdd
      local a 2           ' xdd level
```

the 1st *xdd* is convoluted with a Gaussian with a FWHM of 1 and the 2nd with a Gaussian with a FWHM of 2. In other words, the 1st *gauss_fwhm* equation uses the 'a' parameter from the top level and the second *gauss_fwhm* equation uses the 'a' parameter defined in the 2nd *xdd*. This is analogous, for example, to the scoping rules found in the c programming language. The following is not valid as b1 is defined twice but in a different manner.

```
xdd    local a 1 prm b1 = a;
xdd    local a 2 prm b1 = a;
```

The following comprises 4 separate parameters and is valid:

```
xdd    local a 1 local b1 = a;
xdd    local a 2 local b1 = a;
```

2.5..... Reporting on equation values

When an equation is used in place of a parameter 'name' and 'value' as in

```
     occ Ti+4 = 1-zr;
```

then it is possible to obtain the value of the equation by placing " : 0" after it, or,

```
     occ Ti+4 = 1-zr; : 0
```

After refinement the '0' is replaced by the value of the equation. The associated error is also reported when *do_errors* is defined.

2.6..... Naming of equations

Equations can be given a parameter name, for example:

```
     prm !a1 = a2 + a3/2; : 0
```

Here the a1 parameter represents the equation "a2 + a3/2". If the value of the equation evaluates to a constant then a1 would be an independent parameter, otherwise a1 is treated as a dependent parameter. If the equation evaluates to a constant, then a1 will be refined if the character '!' is not used. The following equation is valid even though it doesn't have a parameter name; its value and error are also reported on termination of refinement.

```
     prm = 2 a1^2 + 3; : 0
```

Equations in general are not evaluated sequentially, the following:

```
     prm a2 = 2 a1; : 0
     prm a1 = 3;
```

gives on termination of refinement:

```
     prm a2 = 2 a1; : 6
     prm a1 = 3;
```

Non-sequential equation evaluation is possible as parameters cannot be defined more than once with different values or equations; the following examples lead to redefinition errors:

```
     prm a1 = 2;    prm a1 = 3;  ' redefinition error
     prm b1 = 2 b3;  prm b1 = b3; ' redefinition error
```

2.7..... existing_prm

[*existing_prm* E]...

Evaluated sequentially and allows for the modification of an existing *prm*/*local* parameters, see for example the macro *K_Factor_WP* in TOPAS.INC. The following:

```
     local a 1
     existing_prm a += 1;
     existing_prm a /= 2;
     existing_prm a = 3 (a + 1);
```

```
    prm = a; : 0
```

will give:

```
    prm = a; : 6.00000
```

Allowed operators for *existing_prm* are +=, -=, *-, /= and ^=.

2.8..... String_To_Variable and Double_To_String functions

The function *Double_To_String* converts a number to a string. The macro *String_To_Variable* converts a string to a variable. Together these macros provide flexibility in the creation of INP files where the numbers of data files and phases are large; here's an example usage:

```
prm cs_L_1 100
prm cs_L_2 100
prm cs_G_1 100
prm cs_G_2 100
xdd …
    str … local str_ 1
    str … local str_ 2
xdd …
    str … local str_ 1
    str … local str_ 2
for xdds {
    for strs {
        lor_fwhm   = 1 / String_To_Prm("cs_L_", Double_To_String(str_));
        gauss_fwhm = 1 / String_To_Prm("cs_G_", Double_To_String(str_));
    }
}
```

In the above, the local parameters str_ acts like structure type identifiers. The *String_To_Variable* function can take any number of strings which are concatenated to form a parameter name. The *Double_To_String* takes one parameter which can be either a constant or a variable.

2.9..... dummy and dummy_prm keywords

The *dummy* keyword reads a word from the input stream. *dummy_prm* is similar except it reads parameter dependent text. For example, the following text in purple is loaded by *dummy_prm* and is ignored by the Kernel.

```
load xo dummy_prm I
    {
        10    = 1/Max(0.00023, 0.0001); min 10 max = Val 2; @ 100
        ...
```

2.10 ... Parameter errors and correlation matrix

When *do_errors* is defined, parameter errors and the correlation matrix are generated at the end of refinement, see also section 4.8. Errors are reported following parameter values as follows:

```
      a lp 5.4031_0.0012
```

Here the error in the lp parameter is 0.0012. The correlation matrix is identified by *C_matrix_normalized* and is appended to the OUT file, if it does not already exist, or updated if it does exist.

2.11 ... Default parameter limits and LIMIT_MIN / LIMIT_MAX

Parameters with internal default *min*/*max* attributes are shown in Table 2-1. These limits avoid invalid numerical operations and equally important they stabilize refinement by directing the minimization process towards lower χ^2 values. Hard limits are avoided where possible and instead parameter values move within a range during a refinement iteration. Refinement without limits sometimes fail in reaching a low χ^2. User defined *min*/*max* limits override default limits; parameters defined using *prm*/*local* should also be defined with user defined *min*/*max* limits. Functionality is often realized through standard macros as defined in TOPAS.INC; this is an important file to view. Almost all *prm*'s defined within this file have associated limits. For example, the *CS_L* macro defines a crystallite size parameter with a *min*/*max* of 0.3 and 10000 nm respectively. On termination of refinement, independent parameters that refined close to their limits are identified by the text "_LIMIT_MIN_#" or "_LIMIT_MAX_#" appended to the parameter value. The '#' corresponds to the limiting value. These warnings can be suppressed using *no_LIMIT_warnings*.

Table 2-1. Default parameter limits.

Parameter	min	max
la	1e–5	2 *Val* + 0.1
lo	Max(0.01, *Val*-0.01)	Min(100, *Val*+0.01)
lh, *lg*	0.001	5
a, *b*, *c*	Max(1.5, 0.995 *Val* – 0.05)	1.005 *Val* + 0.05
al, *be*, *ga*	Max(1.5, *Val* – 0.2)	*Val* + 0.2
scale	1e–11	
sh_Cij_prm	–2 Abs(*Val*) – 0.1	2 Abs(*Val*) + 0.1
occ	0	2 *Val* + 1
beq	Max(–10, *Val*-10)	Min(20, *Val*+10)
pv_fwhm, *h1*, *h2*, *spv_h1*, *spv_h2*	1e–6	2 Val + 20 *Peak_Calculation_Step*
pv_lor, *spv_l1*, *spv_l2*	0	1
m1, *m2*	0.75	30
d	1e–6	
xo	Max(*X1*, *Val* – 40 *Peak_Calculation_Step*)	Min(*X2*, *Val* + 40 *Peak_Calculation_Step*)
l	1e–11	
z_matrix distance	Max(0.5, *Val* .5)	2 *Val*
z_matrix angles	*Val* – 90	*Val* + 90
rotate	*Val* – 180	*Val* + 180
x, *ta*, *qa*, *ua*	*Val* – 1/Get(a)	*Val* + 1/Get(a)

y, *tb*, *qb*, *ub*	*Val* – 1/Get(b)	*Val* + 1/Get(b)
z, *tc*, *qc*, *uc*	*Val* – 1/Get(c)	*Val* + 1/Get(c)
u11, *u22*, *u33*	*Val* If(*Val* < 0, 2, 0.5) – 0.05	*Val* If(*Val* < 0,0.5,2) + 0.05
u12, *u13*, *u23*	*Val* If(*Val* < 0, 2, 0.5) – 0.025	*Val* If(*Val* < 0,0.5,2) + 0.025
filament_length	0.0001	2 *Val* + 1
sample_length, *receiving_slit_length*, *primary_soller_angle*, *secondary_soller_angle*		

2.12 … Reserved parameter names

Table 2-2 and Table 2-4 lists reserved parameter names that are internally updated when needed. Table 2-3 details dependences for certain reserved parameter names. An exception is thrown when a reserved parameter name is used for a User defined parameter name. An example for *weighting* using the reserved parameter names of *Yobs*, *Ycalc* and *X* is as follows:

```
weighting = Abs(Yobs-Ycalc) / (Max(Yobs+Ycalc,1) Max(Yobs,1) Sin(X Deg / 2));
```

Table 2-2. Reserved parameter names.

Name	Description
A_star, *B_star*, *C_star*	Corresponds to the lengths of the reciprocal lattice vectors.
Change	Returns the change in a parameter at the end of a refinement iteration. *Change* can only appear in the equations *update* and *stop_when*.
D_spacing	Corresponds to the d-spacing of phase peaks in Å.
H, *K*, *L*, *M*	hkl and multiplicity of phase peaks.
Iter, *Cycle*, *Cycle_Iter*	Returns the current iteration, the current cycle and the current iteration within the current cycle respectively. Can be used in all equations.
Lam	Corresponds to the wavelength *lo* of the emission profile line with the largest *la* value.
Lpa, *Lpb*, *Lpc*	Corresponds to the *a*, *b* and *c* lattice parameters respectively.
Mi	An iterator used for multiplicities. See the *PO* macro of TOPAS.INC for an example of its use.
Peak_Calculation_Step	Return the calculation step for phase peaks, see *x_calculation_step*.
QR_Removed, *QR_Num_Times_Consecutively_Small*	Can be used in the *quick_refine_remove* equation.
R, *Ri*	The distance between two sites *R* and an iterator *Ri*. Used in the equation part of *atomic_interaction*, *box_interaction* and *grs_interaction*.

Rp, Rs	Primary and secondary diffractometer radius respectively.
T	Corresponds to the current temperature, can be used in all equations.
Th	Corresponds to the Bragg angle (in radians) of hkl peaks.
X, X1, X2	Corresponds to the measured x-axis, the start and the end of the x-axis respectively. *X* is used in *fit_obj*'s equations and the weighting equation. *X1* and *X2* can be used in all *xdd* dependent equation.
Xo	Corresponds to the current peak position; this corresponds to 2Th degrees for x-ray data.
Val	Returns the value of the corresponding parameter.
Yobs, Ycalc, SigmaYobs	Observed, Calculated and estimated standard deviation in *Yobs*; can be used in the weighting equation.

Table 2-3. Parameters that operate on phase peaks; dependencies are not shown.

Keywords that can be a function of *H, K, L, M, Xo, Th* and *D_spacing*.

lor_fwhm	*user_defined_convolution*	*phase_out*, *phase_out_X*
gauss_fwhm	*th2_offset*	*scale_top_peak*
hat	*scale_pks*	*set_top_peak_area*
one_on_x_conv	*h1, h2, m1, m2*	
exp_conv_const	*spv_h1, spv_h2, spv_l1, spv_l2*	*ymin_on_ymax*
circles_conv	*pv_lor, pv_fwhm*	*la, lo, lh, lg*
stacked_hats_conv	*pk_xo*	*modify_peak_eqn*
		current_peak_min_x
		current_peak_max_x

Table 2-4. Phase intensity reserved parameter names.

Name	Description
A01, A11, B01, B11	Used for reporting structure factor details as defined in equations (17–5a) and (17–5b), see the macros *Out_F2_Details* and *Out_A01_A11_B01_B11*.
Iobs_no_scale_pks *Iobs_no_scale_pks_err*	Returns the observed integrated intensity of a phase peak and its associated error without any *scale_pks* applied. *Iobs_no_scale_pks* for phase peak *p* is calculated using the Rietveld decomposition formulae, or, $$^{1}Iobs\_no\_scale\_pks = \text{Get}(scale)\frac{I_P \sum_x P_{x,p} Y_{obs,x}}{Y_{calc,x}}$$ where $P_{x,p}$ is the phase peak *p* calculated at the x-axis position *x*. The summation $\sum_x$ extends over the x-axis extent of the peak *p*. A good fit to the observed data results in an *Iobs_no_scale_pks* being approximately equal to *I_no_scale_pks*.
I_no_scale_pks	The Integrated intensity without *scale_pks* equations, or,

	$^1I\_no\_scale\_pks$ = Get(*scale*) *I*
I_after_scale_pks	The Integrated intensity with *scale_pks* equations applied.
	$^1I\_after\_scale\_pks$ = Get(*scale*) *I* Get(*all_scale_pks*)
	returns the cumulative value of all *scale_pks* equations applied to a phase.

^{1}) *I* corresponds to *I* of *hkl_ls*, *xo_ls* and *d_ls* phases or ($M\,F_{obs}^2$) for *str* phases.

2.13 ... Val and Change reserved parameter names

Val is a reserved parameter name corresponding to the numeric value of a parameter during refinement. *Change* is a reserved parameter name corresponding to the change of a parameter at the end of an iteration as determined by non-linear least squares. *Val* can only be used in the attribute equations *min*, *max*, *del*, *update*, *stop_when* and *val_on_continue*. *Change* can only be used in the attribute equations *update* and *stop_when*. Here are some examples:

```
min 0.0001
max = 100;
max = 2 Val + 0.1;
del = Val 0.1 + 0.1;
update = Val + Rand(0,1) Change;
stop_when = Abs(Change) < 0.000001;
val_on_continue = Val + Rand(-Pi, Pi);
x @ 0.1234 update = Val + 0.1 ArcTan(Change 10); min=Val-.2; max=Val+.2;
```

2.14 ... Using local to assist in using "for ... {}" loops

The following parameters have global scope allowing for use in *for* loops:

```
march_dollase $Name
spherical_harmonics_hkl $Name
sites_geometry $Name
sites_distance $Name
sites_angle $Name
sites_flatten $Name
```

The *march_dollase* parameter, as used in the *PO* macro, can be constrained to the same value across two or more structures by giving them the same name, for example:

```
PO(po1, 0.8, , 1 0 4)
```

This allows the use of '*for xdds* { *for strs* { ... }}'; see PO-CONSTRAINED-CREATE.INP and PO-FOR.INP in the TEST_EXAMPLES\PO-CONSTRAINED directory. Note the use of '*if* Prm_Then(...) { ... }' rather than '*for strs* 1 to 1 { ... }' to facilitate the writing of the INP file. The $Name in *spherical_harmonics_hkl* is *local* but the spherical harmonics coefficients are global. In the following:

```
PO_Spherical_Harmonics(sh2, 8 load sh_Cij_prm {
```

```
        k00    !sh2_c00   1.0000
        k41    sh2_c41    0.1000
        k61    sh2_c61   -0.2000
        k62    sh2_c62    0.3000
        k81    sh2_c81   -0.4000
     } )
```

the sh2 parameter is local to the *str* and the coefficients k00, k41 etc... are global. This allows the constraining of coefficients across different structures within '*for strs*'; see POSH-CON-STRAINED-CREATE.INP and POSH-FOR.INP in the TEST_EXAMPLES\PO-CONSTRAINED directory:

2.15 ... out_dependences and out_dependences_for

[*out_dependences* $user_string]
[*out_dependences_for* $user_string $object_name]

out_dependences outputs dependences for the most previously defined *prm* or *local*. For example, the following:

```
        iters 1
        prm d 1
        prm e 1
        prm f 1
        prm c = e + f;
        prm b = d + e;
        prm a = b + c;
        out_dependences a_tag
        penalty = a^2;
```

produces on refinement termination the following in standard output:

```
    out_dependences a_tag prm_10
       Object name followed by prm name
          prm_10    e
          prm_10    f
          prm_10    d
```

out_dependents_for is similar except that it names an object that is not a parameter, for example, the following lists independent refined parameters associated with the most recently defined *rigid* body:

```
     rigid ...   out_dependents_for tag_1 rigid
```

Many $object_name's can be tagged, these include *x*, *y*, *z*, *occ*, *beq*, *u11*, *u22*, *u33*, *u12*, *u13*, *u23*, *a*, *b*, *c*, *al*, *be*, *ga*, etc. In addition, non-parameters can be tagged, these include *site*, *rigid*, *sites_restrain*, *lat_prms*, *gauss_conv*, *lor_conv*, *all_scale_pks*, *th2_offset_eqn* etc.

2.16 ... The num_runs keyword and preprocessor specifics

[*num_runs* #]
[*out_file* = $E]
[*system_before_save_OUT* { $system_commands }]...
[*system_after_save_OUT* { $system_commands }]...

Typically, an INP file is run once; *num_runs* change's this behavior where the refinement is re-started and performed again until it is performed *num_runs* times. Information from one run to the next can be exchanged via the *out* keyword and the *#include* directive. The INP file is read each Run but not updated when *num_runs* > 1 and *out_file* is empty. Equations during a Run could simplify to a constant, or indeed, the Constant function can be used such that during a Run a parameter is not refined. From TA.EXE and Launch mode the Rwp graphical plot is appended such that it looks like *continue_after_convergence.* The following INP segment:

```
num_runs 10
yobs_eqn aac##Run_Number##.xy = Gauss(Run_Number, 1 + Run_Number);
    min -2 max 20 del 0.01
```

produces on execution the following:
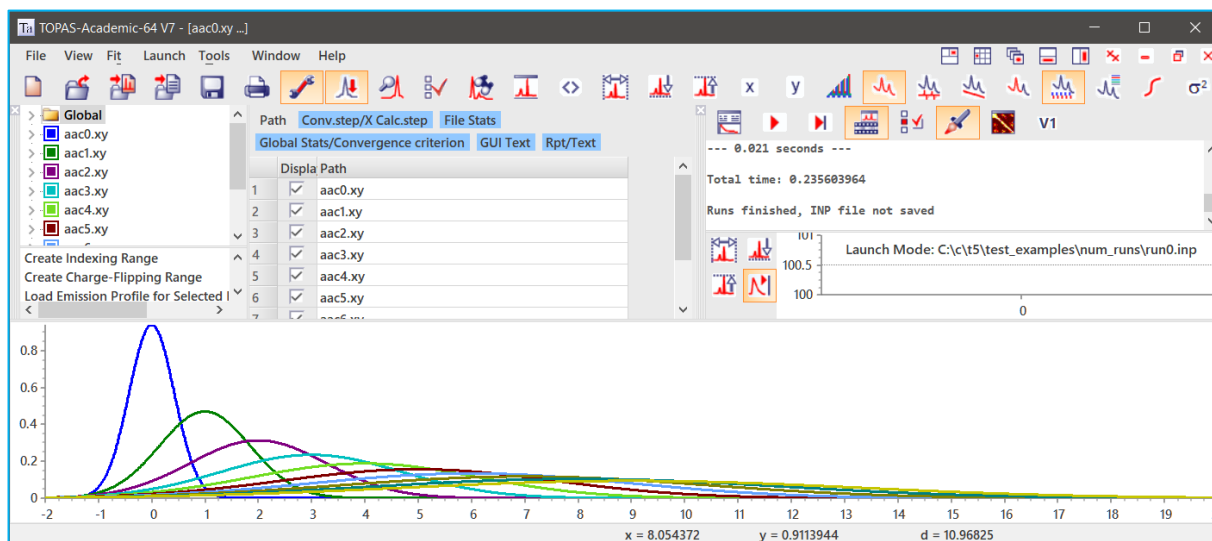
out_file determines the name of the output file updated on refinement termination. The OUT file comprises the INP file but with parameter values updated. *out_file* defaults to the name of the INP file but with an OUT extension. If *num_runs* is greater than 1 and *out_file* is not defined then no OUT file is saved. This can speed up some refinements when an OUT file is not needed. *out_file* is an equation that needs to evaluate to a string; here are some examples:

```
out_file    aac.out  ' This will throw an exception
out_file = aac.out;  ' This will throw an exception
out_file = "aac.out";
out_file = String(aac.out);
out_file = If(Get(r_wp) < 10, "aac.out", "");
out_file = If(Get(r_wp) < 10, Concat(String(INP_File), ".OUT"), "");
```

The standard macro *Save_Best* uses *out_file* as follows:

```
macro Save_Best {
  #if (Run_Number == 0)
     prm Best_Rwp_ = 9999;
  #else
     prm Best_Rwp_ = #include Best_Rwp_.txt;
  #endif
  out Best_Rwp_.txt Out(If(Get(r_wp) < Best_Rwp_, Get(r_wp), Best_Rwp_))
  out_file = If(Get(r_wp) < Best_Rwp_, Concat(String(INP_File),".OUT"), "");
}
```

system_before_save_OUT executes system commands defined in $system_commands string just before the *.OUT file is updated. The system commands are executed from the directory of the INP file. $system_commands can comprise any operating system commands. The macro *Backup_INP* uses *system_before_save_OUT*; it's defined in TOPAS.INC as:

```
macro Backup_INP {
    system_before_save_OUT {
        copy INP_File##.inp INP_File##.backup
    }
}
```

system_after_save_OUT executes the system commands defined in $system_commands string just after the *.OUT file is updated.

2.16.1 .... Reserved macro names

The following are internally generated macros that can be used in INP files.

ROOT : Returns the root directory of the program.

INP_File : Returns current INP file name without a path or extension.

Run_Number : Returns the current run number.

File_Can_Open($file) : Returns 1 if $file can be opened or 0 of it can't be opened.

Running an INP file called AAC.INP from TC.EXE where AAC.INP comprises:

```
ROOT INP_File Run_Number File_Can_Open(aac.xy)
```

and AAC.XY exists will produce in TC.LOG the following:

```
c:\topas-6\ aac 0 1
```

2.16.2 .... The #list directive – creating arrays of macros

#list creates arrays of macros than can be expanded depending on the value of an implied argument. For example, the following creates three arrays of macros called *File_Name*, *Temperature* and *Time*.

```
    #list File_Name & Temperature(, & la) Time {
        File0001.xy 300 0.0
        { File0002 .xy } 320 10.2      ' Line with curly brackets
        File0003.xy 340 21.0
        File0017.xy { 360 + la } 28.9 ' Line with curly brackets
        File0107.xy 380 101.2 }
```

The actual macro invoked depends on the first argument of the macro. In the case of *File_Name* and *Time* the first argument is implied. In the case of *Temperature*, the first argument is the implied argument. When the macro is invoked the first argument is a #type equation that must equate to an integer; here's an example for *File_Name*:

```
    xdd File_Name(Run_Number)
```

Curly brackets, as seen in the above *#list*, can be used as delimiters; the following:

```
    File_Name(1)
    Temperature(1,)
    Temperature(3, Get(la) + 0.01)
```

produces on expansion:

```
    File0002 .xy
    (320)
    (360 + (Get(la) + 0.01))
```

Using curly brackets as delimiters allows for curly brackets themselves to be part of the macro body.

2.16.3 ....The File_Variable and File_Variables macro

The *File_Variable* macro can be used to run a series of runs with parameters changing in a user defined manner between runs; the macro is defined in TOPAS.INC as follows:

```
    macro File_Variable(c, x_start, dx) {
        #if (Run_Number == 0)
            #prm c = x_start;
        #else
            #prm c = #include c##.txt;
        #endif
        #prm c##_next = c + dx;
        out c##.txt Out(#out c##_next)
    }
```

Using *File_Variable* as follows:

```
    File_Variable(occ, 0, 0.1)
```

will generate a file called OCC.TXT for each Run with values ranging from 0.1 to 1 in steps of 0.1. A *#prm* is defined each run with the corresponding values. *#out* can be used to place the *#prm* in the INP file, for example, the following:

```
iters 0
num_runs 11
File_Variable(occ, 0, 0.1)
macro Out_File { Occ##Run_Number##.Out }
out_file Out_File
system_after_save_OUT {
   #if (Run_Number)
      type Out_File >> aac.out
   #else
      type Out_File > aac.out
   #endif
}
yobs_eqn !aac.xy = 1;
   min 10 max 50 del 0.01
   CuKa1(0.0001)
   Out_X_Ycalc( occ##Run_Number##.xy )
   STR(F_M_3_M)
      scale @ 0.0014503208
      Cubic(5.41)
      site  Ce1                          occ Ce+4 = #out occ; beq  0.2028
      site  O1   x 0.25 y 0.25 z 0.25  occ O-2  1           beq  0.5959
```

results in eleven *.XY files each generated with a different occupancy for the Ce1 site as determined by the occ *#prm*. The names of the files would be OCC0.XY to OCC10.XY. Additionally, using *system_after_save_OUT* the file AAC.OUT will contain a concatenation of all the *.OUT files. To iterate over two variables, pa and pb say, then the *File_Variables* macro, defined in TOPAS.INC as:

```
macro File_Variables(a, a1, a2, da, b, b1, b2, db) {
   #if (Run_Number == 0)
      #prm a = a1;
      #prm b = b1;
   #else
      #prm a = #include a##.txt;
      #prm b = #include b##.txt;
   #endif
   #prm a##_next = If(b >= b2, a + da, a);
   #prm b##_next = If(b >= b2, b1, b + db);
   out a##.txt Out(#out a##_next)
   out b##.txt Out(#out b##_next)
}
```

can be used as follows:

```
iters 0
num_runs 36
File_Variables(pa, 0, 1, 0.2, pb, 0, 1, 0.2)
prm !pa = #out pa; prm !pb = #out pb;
out papb.txt append
   out_record out_eqn = pa; out_fmt "(%.1f, "
   out_record out_eqn = pb; out_fmt "%.1f) "
   #if (pb == 1) Out_String("\n") #endif
```

On running the above the PAPB.TXT File contains:

```
(0.0, 0.0) (0.0, 0.2) (0.0, 0.4) (0.0, 0.6) (0.0, 0.8) (0.0, 1.0)
(0.2, 0.0) (0.2, 0.2) (0.2, 0.4) (0.2, 0.6) (0.2, 0.8) (0.2, 1.0)
(0.4, 0.0) (0.4, 0.2) (0.4, 0.4) (0.4, 0.6) (0.4, 0.8) (0.4, 1.0)
(0.6, 0.0) (0.6, 0.2) (0.6, 0.4) (0.6, 0.6) (0.6, 0.8) (0.6, 1.0)
(0.8, 0.0) (0.8, 0.2) (0.8, 0.4) (0.8, 0.6) (0.8, 0.8) (0.8, 1.0)
(1.0, 0.0) (1.0, 0.2) (1.0, 0.4) (1.0, 0.6) (1.0, 0.8) (1.0, 1.0)
```

3. ..EQUATION OPERATORS AND FUNCTIONS

Table 3-1. Operators and functions supported in equations (case sensitive). In addition, equations can be a function of User defined parameter names.

Arithmetic

+, –, *, /	Plus, Minus, Multiply, Divide. Multiply is optional, x*y = x y
^	x^y, Calculates x to the power of y. Precedence:
	$x{\wedge}y{\wedge}z = (x{\wedge}y){\wedge}z, \quad x{\wedge}y^*z = (x{\wedge}y)^*z, \quad x{\wedge}y/z = (x{\wedge}y)/z$

Conditional

a == b	Returns 1 if a = b
a < b	Returns 1 if a < b
a <= b	Returns 1 if a ≤ b
a > b	Returns 1 if a > b
a >= b	Returns 1 if a ≥ b
And(a, b, …)	Returns 1 if all arguments are non-zero
Or(a, b, …)	Returns 1 if one or more argument is non-zero

Mathematical

ArcCos(x)	Returns the arc cos of x (–1 <= x <= 1)
ArcSin(x)	Returns the arc sine of x (–1 <= x <= 1)
ArcTan(x)	Returns the arc tangent of x
ArcTan2(y,x)	Returns arc tangent of y/x
Cos(x)	Returns the cosine of x
Cosh(x)	Hyperbolic cosine
Erf_Approx(x)	Error function
Erfc_Approx	Complementary error function
Exp(x)	Returns the exponential e to the x
Gamma_Approx(x)	Return the Gamma of x
Gamma_Ln_Approx(x)	Returns the natural logarithm of the gamma function
Gamma_P(a, x)	Returns the incomplete Gamma function P(a, x)
Gamma_Q(a, x)	Returns the incomplete Gamma function Q(a, x) = 1-P(a,x)
Ln(x)	Returns the natural logarithm of x
Sin(x)	Returns the sine of x
Sinh(x)	Hyperbolic sine
Sqrt(x)	Returns the positive square root
Tan(x)	Returns the tangent of x
Tanh(x)	Hyperbolic tangent

Special

For(*Mi* = 0, *Mi* < *M*, *Mi* = *Mi*+1 , …)

Get($keyword)	Gets the parameter associated with $keyword
If(conditional_test, return true_eqn, return false_eqn)	
Sum(returns summation_eqn, initializer, conditional_test, increment_eqn)	
Miscellaneous	
Abs(x)	Returns the absolute value of x
Break	Can be used to terminate loops implied by the equations *atomic_interaction*, *box_interaction* and *grs_interaction*.

Break_Cycle	Can be used to terminate a refinement cycle. For example, a refinement cycle can be terminated depending on the value of a penalty as follows:
	
	``
	atomic_interaction ai = (R - 1.3)^2;
	penalty = If(ai > 5, Break_Cycle, 0);
	``

Error(p)	Returns associated error of parameter p.
Max(a,b,c …)	Returns the max of all arguments
Min(a,b,c …)	Returns the min of all arguments
Mod(x, y)	Returns the modulus of x/y. Mod(x, 0) returns 0
Obj_There(a)	Returns 1 if object 'a' exists within the current scope
Prm_There(a)	Returns 1 if prm/local 'a' exists
Rand(a, b)	Returns a uniform deviate random number between a & b
Rand_Normal(mean,stddev)	Returns a random number with a normal distribution with a mean of 'mean' amd standard deviation of 'stddev'
Round(x)	Examples: ``` prm = Round(.1); : 0.00000 prm = Round(.5); : 0.00000 prm = Round(1.6); : 2.00000 prm = Round(-.1); : 0.00000 prm = Round(-.5); : 0.00000 prm = Round(-1.6); : -2.00000 ```
Sign(x)	Returns the sign of x, or zero if x = 0

In addition, the following functions are implemented:

AB_Cyl_Corr(μR), **AL_Cyl_Corr**(μR)

Returns A_B and A_L for cylindrical sample intensity correction (Sabine *et al.*, 1998). These functions are used in the macros *Cylindrical_I_Correction* and *Cylindrical_2Th_Correction*. Example CYLCORR.INP demonstrates usage. For a more accurate alternative to the Sabine corrections see the *capillary_diameter_mm* convolution.

Bkg_at(x)

Returns the value of the Chubychev polynomial, defined by *bkg*, at the value x.

Constant(expression)

Evaluates 'expression' once and then replaces 'Constant(expression)' with the corresponding numeric value. Very useful when the expected change in a parameter insignificantly affects the value of a dependent equation, see for example the *TOF_Exponential* macro.

Get_Prm_Error($prm_name)

Returns the error of the parameter $prm_name.

Ln_Normal_x_at_CD(u, s, v, toll)

Returns x value of a Ln normal distribution such that x is at the Cumulative Distribution value of 'cd' where u and s are the mean and standard deviation of the variable's natural logarithm. x is calculated with a tolerance in 'cd' of 'toll'; see TEST_EXAMPLES\WPPM\LN-NORMAL-1.INP.

PV_Lor_from_GL(gauss_FWHM, lorentzian_FWHM)

Returns the Lorentzian contribution of a pseudo-Voigt approximation to the Voigt where gauss_FWHM and lorentzian_FWHM are the FWHMs of the Gaussian and Lorentzian convoluted to form the Voigt.

Sites_Geometry_Distance($Name)
Sites_Geometry_Angle($Name)
Sites_Geometry_Dihedral_Angle($Name)

Value_at_X(object, x) : Returns the value of *object* at $X = x$. *object* could be a parameter or a *user_y* object. For example, to ensure background is close to the high angle end of a pattern during PDF-generation, the following could be implemented:

```
user_y u capillary.xy
fit_obj = (p0 + p1 X) u;
bkg @ 0 0 0
penalty = 1000 (Bkg_at(X2) + (p0 + p1 X2) Value_at_X(u, X2) - Yobs_at(X2))^2;
```

Voigt_Integral_Breadth_GL(gauss_FWHM, lorentzian_FWHM)

Returns the integral breadth resulting from the convolution of a Gaussian with a Lorentzian with FWHMs of gauss_FWHM and Lorentzian_FWHM respectively.

Voigt_FWHM_GL(gauss_FWHM, lorentzian_FWHM)

Returns the Voigt FWHM resulting from the convolution of a Gaussian with a Lorentzian with FWHMs of gauss_FWHM and Lorentzian_FWHM respectively.

Yobs_Avg(x1, x2)

Returns the average value of *Yobs* between x1 and x2. x1 and x2 is first set to the closest x-axis data point.

Ycalc_at(x)

Returns the value of *Ycalc* at x. Zero is returned if $x < X1$ or $x > X2$.

Yobs_at(#x)

Returns the *Yobs* value at the x-axis position #x; can be used in all sub *xdd* dependent equations.

Yobs_dx_at(#x):

Returns the step size of the observed data at the x-axis position #x; can be used in all sub *xdd* dependent equations. If the step size in the x-axis is equidistant then *Yobs_dx_at* is converted to a constant corresponding to the step size in the data.

Yobs_Min(x1, x2)

Returns the minimum value of *Yobs* between x1 and x2.

3.1..... 'If' and nested 'If' statements

'If' statements can be used in parameter equations, for example:

```
prm a 0.1 prm b 0.1
lor_fwhm = If(Mod(H, 2) == 0, a Tan(Th), b Tan(Th));
```

'If' can also be nested:

```
prm cs 200 update = If(Val < 10, 10, If(Val > 10000, 10000, Val));
```

For those who are familiar with if/else statements, the *IF THEN ELSE ENDIF* macros, as defined in TOPAS.INC, can be used:

```
IF a > b THEN
    a ' return expression value
ELSE
    b ' return expression value
ENDIF
```

Min and Max can be used in equations, for example:

```
prm a 0.1 prm b 0.3
th2_offset = Min(Max(a, b, -0.2), 0.2);
```

3.2..... Floating point exceptions

An exception is thrown when an invalid floating-point operation is encountered, i.e.

Divide by zero

Sqrt(x) for x < 0

Ln(x) for x ≤ 0

ArcCos(x) for x < -1 or x > 1

Exp(x) produces an overflow for x ~> 700

(-x)^y for x > 0 and y not an integer

Tan(x) evaluates to Infinity for x = n Pi/2, Abs(n) = 1, 3, 5, …

min/*max* equations, Min/Max functions or 'If' functions can be used to avoid invalid floating-point operations. Equations can also be manipulated to yield valid floating-point operations, for example, Exp(-1000) can be used in place of 1/Exp(1000).

4. ..THE MINIMIZATION ROUTINES

TMinimization
    [*line_min*][*use_extrapolation*][*no_normal_equations*][*use_LU*]
    [*approximate_A*]
        [*A_matrix_memory_allowed_in_Mbytes* !E]
        [*A_matrix_elements_tollerance* !E]
        [*A_matrix_report_on*]
    [*chi2* !E]
    [*chi2_convergence_criteria* !E]
    [*continue_after_convergence*]
    [*bootstrap_errors* !Ecycles]
        [*fraction_of_yobs_to_resample* !E]
        [*determine_values_from_samples*]
        [*resample_from_current_ycalc*]
    [*do_errors*]
    [*do_errors_include_restraints*]
    [*do_errors_include_penalties*]
    [*only_penalties*]
    [*percent_zeros_before_sparse_A* #]
    [*penalty* !E]...
    [*penalties_weighting_K1* !E]
    [*pen_weight* !E]
    [*quick_refine* !E [*quick_refine_remove* !E]]
    [*randomize_on_errors*]
    [*restraint* !E]
    [*save_best_chi2*]
    [*use_LU_for_errors*]

Get(number_independent_parameters)

The Newton-Raphson non-linear least squares method is used by default with the Marquardt method (1963) included for stability. The objective function χ^2 is written as:

$$\chi^2 = \chi_0^2 + \chi_P^2 + \chi_R^2 \tag{4-1}$$

$$where \quad \chi_0^2 = K \sum_{m=1}^{M} w_m \left(Y_{o,m} - Y_{c,m} \right)^2 \tag{4-2}$$

$$\chi_P^2 = KK_1K_P \sum_{p=1}^{N_p} P_p \qquad \chi_R^2 = KK_1K_R \sum_{r=1}^{N_R} R_r^2 \qquad K = \frac{1}{\sum_{m=1}^{M} w_m Y_{o,m}^2} \tag{4-3}$$

$Y_{o,m}$ and $Y_{c,m}$ are the observed and calculated data respectively at data point m, M the number of data points, w_m the weighting given to data point m which for counting statistics is given by $w_m = 1/\sigma(Y_{o,m})^2$ where $\sigma(Y_{o,m})$ is the error in $Y_{o,m}$, P_p are penalty functions, defined using *penalty*,

and N_p the number of penalty functions. R_r are restraints, defined using *restraint*, and N_R the number of restraints. K_P and K_R are weights applied to the penalty functions and restraints respectively. K_1 corresponds to the user defined *penalties_weighting_K1* (default value of 1), typical values range from 0.1 to 2. Penalty functions and Restraints are minimized when there are no observed data Y_o; see example ONLYPENA.INP.

The matrix equations are generated by the usual expansion of $Y_{c,m}$ to a first order Taylor series around the parameter vector **p**. The size of **p** corresponds to the number of independent parameters N. The penalty functions are expanded to a second order Taylor series around the parameter vector **p**. The restraints are expanded to a first order Taylor series around the parameter vector **p.** The resulting matrix equations are:

$$A \, \Delta \mathbf{p} = Y \qquad (4\text{-}4)$$

$$\text{where} \quad \mathbf{A} = \mathbf{A}_0 + \mathbf{A}_P + \mathbf{A}_R$$

$$\text{and} \quad \mathbf{Y} = \mathbf{Y}_0 + \mathbf{Y}_P + \mathbf{Y}_R$$

$$A_{ij} = \sum_{m=1}^{M} w_m \frac{\partial Y_{c,m}}{\partial p_i} \frac{\partial Y_{c,m}}{\partial p_j} \qquad\qquad Y_{o,i} = \sum_{m=1}^{M} w_m \left(Y_{o,m} - Y_{c,m} \right) \frac{\partial Y_{c,m}}{\partial p_i} \qquad (4\text{-}5)$$

$$A_{P,ij} = \frac{K_P}{2} \sum_{p=1}^{N_P} \frac{\partial^2 P_p}{\partial p_i \partial p_j} \qquad\qquad Y_{P,i} = -\frac{K_P}{2} \sum_{p=1}^{N_P} \frac{\partial P_p}{\partial p_i}$$

$$A_{R,ij} = K_R \sum_{r=1}^{N_R} \frac{\partial R_{r,i}}{\partial p_i} \frac{\partial R_{r,j}}{\partial p_j} \qquad\qquad Y_{R,i} = -K_R \sum_{r=1}^{N_R} R_r \frac{\partial R_r}{\partial p_i}$$

The Taylor coefficients $\Delta\mathbf{p}$ corresponds to changes in the parameters p. Eq. (4-4) represents a linear set of equations in $\Delta\mathbf{p}$ that are solved for each iteration of refinement. Off diagonal terms in $\mathbf{A}_P$ are not calculated and are instead set to zero. K_R and K_P are both set to 1 in the absence of χ_0^2. When χ_0^2 does exist then K_P is used to give approximate equal weights to the sum of the inverse error terms in the parameters, $\sigma_0(p_i)^2$ and $\sigma_P(p_i)^2$, calculated from χ_P^2 and χ_0^2 respectively. Neglecting the off-diagonal terms results in $\sigma_P(p_i)^2 = 1/A_{0,ii}$ and $\sigma_P(p_i)^2 = 1/A_{P,ii}$; however, to avoid numerical stabilities K_P is written as shown in Eq. (4-6).

$$K_P = \sum_{k=1}^{N_P} If\left(Y_{P,k} < 10^{-14} A_{0,kk}, 0, \frac{1.05 A_{0,kk}}{\left(A_{P,k} + A_{0,kk} Min\left(Y_{P,kk}/Y_{o,kk}, 0.05 \right) \right)} \right) \qquad (4\text{-}6)$$

k corresponds to independent parameters that are a function of χ_P^2. Similarly, for K_R we have:

$$K_R = \sum_{k=1}^{N_R} If\left(Y_{R,k} < 10^{-14} A_{0,kk}, 0, \frac{1.05 A_{0,kk}}{\left(A_{R,k} + A_{0,kk} Min\left(Y_{R,kk}/Y_{o,kk}, 0.05 \right) \right)} \right) \qquad (4\text{-}7)$$

K_R and K_P can be modified using *pen_weight* and the macro *Pen_Wt*. *Pen_Wt* calls the user defined macro *Write_Pen_Wt*; a definition that mimics the default is:

```
macro Write_Pen_Wt(Aii, Ai, Pii, Pi) {
    pen_weight = If(Pii < 1e-14 Aii,0,1.05 Aii/(Pii+Aii Min(Pi/Ai, 0.05)));
}
```

A_{ii} and A_i corresponds to $A_{0,ii}$ and $Y_{0,i}$ respectively. For K_P then P_{ii} and P_i corresponds to $A_{P,ii}$ and $Y_{P,i}$. For K_R then P_{ii} and P_i corresponds to $A_{R,ii}$ and $Y_{P,i}$. ShelX type restraints can be formulated as follows:

```
pen_weight = 1;
penalties_weighting_K1 = (Get(r_wp) / Get(r_exp))^2;
do_errors_include_restraints
save_best_chi2
restraint = Sqrt(w) (yt - y);
```

where Sqrt(w) is simply the square root of the restraint weight used by ShelX.

4.1 ..... The Conjugate Gradient Solution method

The Bounds Constrained Conjugate Gradient (BCCG) method (Coelho, 2005) incorporating *min*/*max* limits is used for solving the normal equations; it greatly assists in convergence of the non-linear least squares process. *min*/*max* limits are dynamically recalculated and adhered to during the solution process. For example, to constrain site occupancies on three sites to full occupancy with three atomic species, each with occupancy of 1, then the following could be defined (see TEST_EXAMPLES\OCC-CONSTRAIN.INP):

```
site Ni x 0.11 y 0.22 z 0.33     occ Ni ni1  0.20000 min 0 max  1
                                 occ Zr zr1  0.30000 min 0 max = 1 - ni1;
                                 occ Ca ca1 = 1 - ni1 - zr1; :  0.50000

site Zr x 0.21 y 0.32 z 0.43     occ Ni ni2  0.40000 min 0 max = 1 - ni1;
                                 occ Zr zr2  0.50000 min 0 max = 1 - ni2;
                                 occ Ca ca2 = 1 - ni2 - zr2; :  0.10000

site Ca x 0.31 y 0.42 z 0.53     occ Ni ni3 = 1 - ni1 - ni2; :  0.40000
                                 occ Zr zr3 = 1 - zr1 - zr2; :  0.20000
                                 occ Ca ca3 = 1 - ca1 - ca2; :  0.40000

' Occupancy on sites add up to 1
prm = ni1 + zr1 + ca1; :  1.00000
prm = ni2 + zr2 + ca2; :  1.00000
prm = ni3 + zr3 + ca3; :  1.00000

' Individual species add up to 1
prm = ni1 + ni2 + ni3; :  1.00000
prm = zr1 + zr2 + zr3; :  1.00000
prm = ca1 + ca2 + ca3; :  1.00000
```

If the **A** matrix is not sparse then LU-decomposition can be used instead of the BCCG routine by defining *use_LU*. LU-decomposition does not use *min*/*max* limits during the solution process and in addition it requires the full A matrix which may be too memory intensive for problems with

thousands of parameters. LU-decomposition can also be too slow when the number of parameters is greater than about one thousand. *percent_zeros_before_sparse_A* defines the percentage of the A matrix that can be zero before sparse matrix methods are invoked. The default value is 60%.

4.2..... The Marquardt method

The Marquardt (1963) method applies a scaling factor η, called the Marquardt constant, to the diagonal elements of the **A** matrix when the solution to the normal equations of Eq. (5-4) fails to reduce χ^2, or,

$$A_{ii,scaled} = A_{ii}(1 + \eta)$$

After applying the Marquardt constant, the normal equations are again solved and χ^2 recalculated. If χ^2 increases, then η is increased and the scaling process repeated. Repeated failure results in a very large Marquardt constant; taken to the limit the off-diagonal terms can be ignored and the solution to the normal equations can be approximated as:

$$\Delta p_i = Y_i / (A_{ii}(1 + \eta)) \tag{4-8}$$

Improvements to the automatic determination of the Levenberg-Marquardt constant (ᶜCoelho, 2018) has been made. This is especially the case for objective functions that are far from quadratic and when the BFGS method is used. Large refinements, single crystal protein refinements for example, should see convergence rates increase.

4.3..... Approximating the A matrix - the BFGS method

approximate_A can be used to approximate the **A** matrix, Eq. (4-4), without the need to calculate the **A** matrix dot products. The approximation is based on the BFGS method (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970). Approximating A is useful when the calculation of the **A** matrix dot products is proving too expensive. When penalties dominate a refinement then the use of *approximate_A* may also improve convergence. *approximate_A* cannot be used with *line_min* or *use_extrapolation*. The single crystal refinement examples of AE14-APPROX-A.INP and AE1-APPROX-A.INP are cases where the use of *approximate_A* achieves convergence in less time than with the calculated A matrix. When using *approximate_A* the **A** matrix can be made sparse by defining *A_matrix_memory_allowed_in_Mbytes* and/or *A_matrix_elements_tollerance*. This allows for refinements with large numbers of independent parameters. *A_matrix_memory_allowed_in_Mbytes* limits the memory used by the **A** matrix. *A_matrix_elements_tollerance* #tol removes elements in the **A** matrix with values less than #tol. The comparison is made againts normalized elements of **A** such that A_{ii}=1. Typical values for #tol range from 0.0001 to 0.01. *A_matrix_memory_allowed_in_Mbytes* and *A_matrix_elements_tollerance* can be used simultanuously. *A_matrix_report_on* displays the percentage of non-zero elements in the **A** matrix.

4.4..... Line minimization and Parameter extrapolation

Line minimization, also known as the steepest decent method, is invoked using *line_min*. It uses a direction in parameter space given by $\Delta p_i = Y_i/A_{ii}$ to minimize on $\chi^2(p + \lambda \Delta p)$ by adjusting λ. Parameter Extrapolation, *use_extrapolation*, uses parabolic extrapolation of the parameters as a

function of iteration, or, λ is adjusted such that $\chi^2\left(\mathbf{a}\lambda^2 + \mathbf{b}\lambda + \mathbf{c}\right)$ is minimized where for a particular parameter p_i at iteration k we have $a_i = (y_1 - 2y_2 + y_3)/2$, $b_i = (y_3 - y_1)/2$ and $c_i = y_2$ where $y_1 = (p_{i,k-5} + p_{i,k-4})/2$, $y_2 = (p_{i,k-3} + p_{i,k-2})/2$ and $y_3 = (p_{i,k-1} + p_{i,k-0})/2$. Parameter Extrapolation encompasses the last six sets of parameter values. In cases where both exists then Parameter Extrapolation reduces possible oscillatory behavior in χ_0^2 and χ_P^2. Parameter extrapolation when used with Line Minimization can increase the rate of convergence when refining on penalties only. Line minimization and Parameter Extrapolation have relatively small memory footprints and thus can be useful when the **A** matrix consumes too much memory. Alternatively, *approximate_A* can be used. Line minimization with the full **A** matrix calculation (*approximate_A* not defined) can increase the rate of convergence on problems like Pawley refinement. *no_normal_equations* prevent's the use of normal equations in the minimization routine; it is useful when the effects of line minimization are sought or when there's a large number of independent parameters.

4.5..... Restraints and Penalties

penalty defines a penalty function that can be a function of parameters. Penalties such as bond-length restraints are useful for stabilizing refinements. Example HOCK.INP uses penalties to minimize on the Hock and Schittkowski problem number 65:

```
prm x1 1 min -4.5 max 4.5  val_on_continue = Rand(-4.5, 4.5); del 0.01
prm x2 1 min -4.5 max 4.5  val_on_continue = Rand(-4.5, 4.5); del 0.01
prm x3 1 min -5.0 max 5.0  val_on_continue = Rand(-5.0, 5.0); del 0.01

' Hock and Schittkowski problem number 65 function
penalty = (x1 - x2)^2 + (1/9) (x1 + x2 - 10)^2 + (x3 - 5)^2; : 0

prm contraint_1 = x1^2 + x2^2 + x3^2;
penalty = If(contraint_1 < 48, 0, (contraint_1 - 48)^2); : 0
```

To apply a penalty function to lattice and crystallite size parameters, which are expected to be 5.41011 Å and 200 nm respectively, the following can be used:

```
str
    Cubic(lp_ceo2 5.41011)
    CS_L(cs_l, 200)
    penalty = (lp_ceo2 - 5.41011)^2;
    penalty = (cs_l - 200)^2;
```

penalties_weighting_K1 defines the weighting K_1 in Eq. (4-3); the default value is 1. A *restraint* can be reformulated using a penalty by squaring the restraint, for example:

```
restraint = a (x - b);
```

This is equivalent to:

```
penalty = a^2 (x - b)^2;
```

In the case of the *restraint*, the off-diagonal terms $A_{R,ij}$ are calculated when *approximate_A* (the BFGS method) is not defined. In the case of the *penalty* the off-diagonal terms $A_{P,ij}$ is set to zero. Restraints often converge in less iterations than equivalent penalties due to the use of the off-

diagonal terms (compare ROSENBROCK-10.INP with ROSENBROCK-10-RESTRAINT.INP). However, the time to convergence may be greater due to calculation of the off-diagonal restraint terms in the **A** matrix. Penalties are useful for functions that are not to be squared; these include negative functions such as the GRS series atomic interaction (see ALVO4-GRS-AUTO.INP). For efficiency the **A**$_R$ matrix is treated as a sparse matrix which is combined with **A**$_0$ (if it exists) where **A**$_0$ could be either sparse or dense. When *approximate_A* is used then the off-diagonal elements of **A**$_0$, **A**$_P$, and **A**$_R$ are not calculated; instead they are approximated by the BFGS method. When *approximate_A* is used together with penalties and restraints then this effectively means that the restraints are treated as penalties. The following two cases will have similar but not identical convergence.


```
' Case 1

approximate_A
prm p1 1 prm r1 1
penalty !P1 = 5^2 (p1 - 7)^2;
penalty !P2 = 6^2 (p1 - 8)^2;
restraint !R1 = 7 (r1 - 9);
restraint !R2 = 8 (r1 - 10);
```

```
' Case 2

prm p1 1 prm r1 1
penalty !P1 = 5^2 (p1 - 7)^2;
penalty !P2 = 6^2 (p1 - 8)^2;
penalty !P3 = 7^2 (r1 - 9)^2;
penalty !P4 = 8^2 (r1 - 10)^2;
```

Diagonal elements of the A matrix

$A_{P,p1p1} = (½)\partial^2(P1+P2)/\partial p1^2$	$A_{P,p1p1} = (½)\partial^2(P1+P2)/\partial p1^2$
$A_{R,r1r1} = (\partial R1/\partial r1)^2 + (\partial R2/\partial r1)^2$	$A_{P,r1r1} = (½)\,\partial^2(R1^2+R2^2)/\partial r1^2$

The difference in behaviour between penalties and restraints can be seen by comparing ROSENBROCK-10.INP to ROSENBROCK-10-RESTRAINT.INP. In 500,000 iterations, the former results in 71 iterations on average to convergence and the latter 47 iterations on average to convergence. The restraints case converges faster as $A_{R,ij}$ elements are calculated. Approximating $A_{R,ij}$ by defining *approximate_A* in ROSENBROCK-10-RESTRAINT.INP gives the fastest convergence time wise with 71 iterations on average to convergence. Many penalties however cannot be formulated as a restraint, RASTRGIN.INP for example, and in these cases, penalties are necessary.

4.6 ..... Minimizing on penalties only

When there are no observed data or when *only_penalties* is defined then by default the BFGS method is used, see examples ROSENBROCK-10.INP and HOCK.INP. For 'penalties only' the BFGS method typically converges faster than *line_min*/*use_extrapolation*; this however, for 'penalties only', can be overridden with the use of *line_min*.

4.7 ..... Saved refined values and save_best_chi2

Values saved on termination of refinement are determined as follows:

- If *continue_after_convergence* is NOT defined and *save_best_chi2* is NOT defined then values saved corresponds to those of the last iteration.

- If *continue_after_convergence* is NOT defined and *save_best_chi2* is defined then values saved corresponds to those that gave the best χ^2.

- If *continue_after_convergence* is defined and *save_best_chi2* is NOT defined then values saved corresponds to those that gave the best Rwp.

- If *continue_after_convergence* is defined and *save_best_chi2* is defined then values saved corresponds to those that gave the best χ^2.

When there are no penalties or restraints then the best χ^2 corresponds to the best Rwp.

4.8..... Error calculation

Estimated standard deviations for refined independent parameters are calculated at the end of refinement. The correlation matrix, if defined using *C_matrix_normalized*, is updated, if not defined then *C_matrix_normalized* is created and appeneed to the OUT file.

do_errors: Errors calculated; penalties and restraints NOT included in the **A** matrix.

do_errors_include_restraints: Errors calculated; restraints included in the **A** matrix.

do_errors_include_penalties: Errors calculated; penalties included in the **A** matrix.

4.9..... Error determination using SVD and bootstrap errors

Errors have previously been determined from a covariance matrix obtained by LU decomposition. Version 6 onwards use Singular Value Decomposition (SVD) with resulting errors typically smaller for strongly correlated parameters. Additionally, SVD errors more closely resemble those obtained by the boot strap method. *bootstrap_errors* are potentially more accurate as parameter limits are considered; for example, the fact that intensities are positive is not considered by matrix inversion. *use_LU_for_errors* forces the use of LU decomposition; LU-decomposition results in very large errors for intensities that are 100% correlated. The three means of determining errors are demonstrated in a Pawley refinement of Y_2O_3 in the examples Y2O3A-LU.INP, Y2O3A-SVD.INP and Y2O3A-BOOT.INP found in the directory TEST_EXAMPLES\SVD-ERRORS.

bootstrap_errors uses the bootstrap method of error determination (Efron & Tibshirani 1986, DiCiccio & Efron 1996, Chernick 1999). Bootstrapping comprises a series of refinements each with a fraction of *Yobs* modified to obtain a new bootstrap sample. The standard deviations of the refined values then become the bootstrap errors. !Ecycles corresponds to the number of refinement cycles to perform for bootstrapping, it defaults to 200. The resulting bootstrap errors are written to the *.OUT file. *fraction_of_yobs_to_resample* corresponds to the fraction of the observed data that is to be replaced each refinement cycle, it defaults to 0.37. Replacement data is by default obtained randomly from the calculated pattern obtained at the end of the first refinement cycle. If *resample_from_current_ycalc* is defined, then replacement data are obtained from the currently completed refinement cycle. The updated *Yobs* data is additionally modified such that the change in Rwp is unchanged with respect to the current *Ycalc*. Parameter values used at the start of each refinement cycle are obtained from the end of the first refinement cycle. *val_on_continue* can additionlly be used to change parameter values at the start of a cycle. If *determine_values_from_samples* is defined then parameter values at the end of bootstrapping are updated with values determined from the bootstrapping refinement cycles. Parameter values obtained at the end of each bootstrap refinement cycle is written to disk in binary format. These

values are then read and processed at the end of the bootstrap process without storing the values in memory; the bootstrap process therefore has a small memory footprint.

4.10 … Error Propagation using **prm_with_error**

Parameter errors determined outside of refinement can be included and propagated to dependent parameters using *prm_with_error*. For example, consider the INP snippet (see TEST_EXAMPLE\PRM-WITH-ERROR.INP):

```
xo_Is
    xo 0 I = 10 t i;
    prm i  9.99999`_0.00065 min 1e-6
    prm_with_error !t  1_0.33
    prm t_squared = t^2; : 1.00000`_0.66000
```

Here *t* is defined using *prm_with_error* and with an error of 0.33; this error is used to determine errors for dependent parameters, such as t_squared, that are a function of *t*.

4.11 … xdd_sum and xdd_array

[*xdd_array* !E] …	**Example**
[*xdd_sum* !E] …	TEST_EXAMPLES\PDF\GENERATE\I15-DECON.INP

xdd_array calculates and stores an array of values which can then be used in equations which can in turn be a function of the reserved parameter names of *X*, *Yobs*, *Ycalc* and *SigmaYobs*. For example, applying the Si atomic scattering factor correction to an *xo_Is* phase can be performed as follows:

```
xo_Is …
    xdd_array si_f0 =
        2 ( ' atomic scattering data from atmscat.cpp
        5.275329 Exp( -2.631338  (Sin(X Pi/360)/Lam)^2 ) +
        3.191038 Exp( -33.730728 (Sin(X Pi/360)/Lam)^2 ) +
        1.511514 Exp( -0.081119  (Sin(X Pi/360)/Lam)^2 ) +
        1.356849 Exp( -86.288643 (Sin(X Pi/360)/Lam)^2 ) +
        2.519114 Exp( -1.170087  (Sin(X Pi/360)/Lam)^2 ) +
        0.145073);
    scale_phase_X = si_f0; ' apply the atomic scatter factor
```

The above will give the same result if *xdd_array* is replaced by *prm*. The latter does not store the array and therefore the equation is calculated every time si_f0 is used. Because *xdd_array* is an equation, the program also automatically keeps track of its dependencies; this means *xdd_array* array is recalculated only when the equation changes; changes can happen, for example, if the equation is a function of a refinable parameter and the refined parameter changes. This recalculation only occurs when the array is being referenced; it does not occur at the instance of a dependency change. Use of *xdd_array* therefore produces fast and efficient INP files.

xdd_sum is similarto *xdd_array* except an array is not stored; instead the sum of the values of the array are calculated and stored. Similarto *xdd_array*, the summed value of *xdd_sum* is only

recalculated when necessary. In Version 7, *xdd_sum* can be nested, for example, to normalize the intensities between *Yobs* and *Ycalc* the following is now possible:

```
xdd_sum sum_yobs  = Yobs;
xdd_sum sum_ycalc = Ycalc;
xdd_sum = (Yobs - Ycalc sum_yobs / sum_ycalc)^2;
xdd_sum num_data_points = 1;: 0 ' 0 is replace by the number of data points
```

4.12 ... Refining on an arbitrary Chi2

chi2 allows for the minimization of a user defined χ^2. It can be a function of the reserved parameter names *X*, *Yobs*, *Ycalc* and *SigmaYobs*. In addition, *xdd_sum* can be a function of these reserved parameter names. To, for example, define a normal least squares refinement the following can be used:

```
xdd ...
    xdd_sum denominator = Yobs;
    xdd_sum numerator = (Yobs - Ycalc)^2 / Max(Yobs, 1);
    chi2 = 100 Sqrt(numerator / denominator);
```

In refining on an arbitrary *chi2* the first and second derivatives of *chi2* with respect to each independent parameter is required. To do this fast, *Ycalc* within *chi2* is approximated with a first order Taylor approximation around the parameter vector **p**. This approximation, for various formulations of *chi2*, has yielded good convergence even for non-linear parameters. To summarize:

- *chi2* is treated as a penalty.
- For each independent parameter, a definite minimum in *chi2* is bracketed; inverse parabolic interpolation is then used to determine the minima of *chi2* with respect to that parameter. In the calculation of *chi2*, *Ycalc* is replaced with its first order Taylor approximation and thus the full *Ycalc* is only calculated once per refinement iteration and not 100s of times.
- Finding the minima and the curvature of *chi2* with respect to each parameter yields 1st and 2nd order derivatives of *chi2* with respect to each parameter.
- The BFGS method (*approximate_A*) is then used to solve the resulting linear equations with off diagonal terms approximated according to the BFGS method.
- The BCCG method incorporating the Marquardt method with automatic Marquardt constant determination is used to solve the matrix equations.

The Rietveld refinement example of TEST_EXAMPLES\CHI2-CEO2.INP demonstrates various scenarios:

Case 1) Here's output when NOT using *chi2*.

```
0  Time   0.05  Rwp   26.630    0.000 MC   0.00 0
1  Time   0.06  Rwp   16.651   -9.979 MC   0.06 1
2  Time   0.06  Rwp    7.510   -9.141 MC   0.02 1
3  Time   0.08  Rwp    6.955   -0.556 MC   0.01 1
4  Time   0.08  Rwp    6.943   -0.011 MC   0.00 1
5  Time   0.08  Rwp    6.923   -0.020 MC   0.00 1
6  Time   0.09  Rwp    6.923   -0.000 MC   0.18 1
 --- 0.094 seconds ---
```

Case 2) Here's output when NOT using *chi2* but using *approximate_A*.

```
 0  Time   0.05  Rwp   26.630    0.000 MC   0.00 0
 1  Time   0.06  Rwp   16.883   -9.747 MC   0.00 0
 ...
16  Time   0.13  Rwp    6.950   -0.002 MC   0.04 1
17  Time   0.14  Rwp    6.949   -0.002 MC   0.09 1
18  Time   0.14  Rwp    6.949   -0.000 MC   0.29 1
--- 0.14 seconds ---
```

Case 3) Here's output using *chi2* defined for normal least squares.

```
 0  Time   0.03  Rwp   26.630    0.000 MC   0.00 0 P   26.63020
 1  Time   0.06  Rwp   15.897  -10.733 MC   0.00 0 P   15.89696
 ...
13  Time   0.33  Rwp    6.974   -0.021 MC   0.00 1 P    6.97366
14  Time   0.34  Rwp    6.958   -0.016 MC   0.00 1 P    6.95755
15  Time   0.38  Rwp    6.951   -0.006 MC   0.00 1 P    6.95122
```

The *chi2* case (3) looks similar-to case (2); however, the path towards the minima is different as the *chi2* procedure is very different to normal least squares refinement.

4.13 ... Informing on unrefined parameters

Parameters that do not take part in refinement are reported, for example, the following:

```
prm a 1 prm b 1
```

where a and b are not used in any equations that are part of the refinement will result in the output:

```
Number of independent parameters not taking part in refinement: 2
   prm_10:  a
   prm_10:  b
```

The *val_on_continue* attribute of unrefined parameters are executed at the end of convergence. It can be useful, for example,

```
prm a 1 val_on_continue = b = 2; ' this sets the b parameter to 2
```

4.14 ... Summary, Iteration and Refinement Cycle

Table 4-1 shows various keyword usages for typical refinement problems. The term "refinement cycle" is used to describe a single convergence. The reserved parameter *Cycle* returns the current refinement cycle with counting starting at zero. The reserved parameter *Cycle_Iter* returns the current iteration within the current *Cycle* with counting starting at zero.

Table 4-1. Keyword sequences for various refinement types.		
Refinement type	**Keywords to use**	**Comments**
Rietveld refinement. No penalties.		Marquardt refinement. **A** matrix calculation.

Rietveld refinement with a moderate number of penalties.	*line_min* (Maybe)	Line minimization used if *line_min*. Marquardt refinement. **A** matrix calculation.
Rietveld refinement dominated by penalties.	*approximate_A*	BFGS method of refinement. **A** matrix approximation.
Pawley refinement.	*line_min*	Line minimization. Marquardt refinement. **A** matrix calculation.
Penalties only.		BFGS method of refinement. **A** matrix approximation.
Refinements with a large number of parameters.	*approximate_A*	BFGS method of refinement. **A** matrix approximation.

4.15 … quick_refine and computational issues

The computationally dominant factor of Eq. (4-5) is problem dependent. For Rietveld refinement with a moderate number of parameters then the calculation of the peak parameter derivatives may well be the most expensive. On the other hand, for Rietveld refinement with a large number of structural parameters and data points then the calculation of the $A_{1,ij}$ dot products would be the dominant factor, where, the number of operations scale by $M(N^2+N)/2$. Before the development of the BCCG routine (Coelho, 2005), the solution to the normal equations, Eq. (4-4), was also very expensive. For structure solution from powder data by simulated annealing, *yobs_to_xo_posn_yobs* can be used to reduce the number of data points M which reduces the number of operations in the $A_{1,ij}$ dot products, see the *Decompose* macro in example CIME-DECOMPOSE.INP. *quick_refine* removes parameters during a refinement cycle thus shrinking the size of the **A** matrix by reducing N and hence speeding up refinement iterations. Parameters are removed if the condition defined in Eq. (4-9) is met for three consecutive iterations.

$$\Delta p_i < 0.01 \; \text{quick\_refine}/(K \, N \, Y_i) \qquad (4-9)$$

Alternatively, parameters can be removed or reinstated during a refinement cycle using *quick_refine_remove*. This keyword provides a means of performing block refining. If *quick_refine_remove* is not defined, then all parameters are reinstated at the start of refinement cycles. Use of *quick_refine* speeds up simulated annealing, see for example the macro *Auto_T* which is used in example AE1-AUTO.INP. All refined parameters are reinstated for refinement at the start of subsequent cycles. Large *quick_refine* values aggressively removes parameters and convergence to low χ^2 maybe hindered. A value of 0.1 works well. *quick_refine* has the following consequences:

- If parameters are not reinstated using *quick_refine_remove* then χ^2 does not get to its lowest possible value for a particular refinement cycle.

- The degree of parameter randomization increases with increasing values of *quick_refine*. Randomization should therefore be reduced as *quick_refine* increases. Alternatively

randomize_on_errros can be used which automatically determines the amount a parameter is randomized.

If *quick_refine_remove* evaluates to a non-zero value then the associated parameter is removed from refinement, similarly parameters are reinstated if *quick_refine_remove* evaluates to zero. *quick_refine_remove* can be a function of the reserved parameters *QR_Removed* or *QR_Num_Times_Consecutively_Small* and additionally global reserved parameter names such as *Cycle_Iter*, *Cycle* and *T*. If *quick_refine_remove* is not defined, then the removal scheme of Eq. (4-9) is used and parameters are not reinstated until the next refinement cycle. In most refinements the following will produce close to the lowest χ^2 and in a short time period (see for example PAW-LEY1.INP).

```
quick_refine 0.1
   quick_refine_remove =
      IF QR_Removed THEN
         0 ' reinstate the parameter
      ELSE
         IF QR_Num_Times_Consecutively_Small > 2 THEN
            1 ' remove the parameter
         ELSE
            0 ' dont remove the parameter
         ENDIF
      ENDIF;
```

4.16 ... Simulated annealing and Auto_T

Refinement is continued after convergence when *continue_after_convergence is defined*. Before continuing the following actions are performed:

- *val_on_continue* equations for independent parameters are evaluated.
- *randomize_on_errors* processes are performed.
- *rand_xyz* processes are performed.

When *val_on_continue* is defined, the corresponding parameter is not randomized according to *randomize_on_errors*. Simulated annealing is invoked using *continue_after_convergence.* It is sometimes difficult to formulate optimum *val_on_continue* functions; this is especially true in structure solution using rigid bodies where optimum randomization of the rigid body parameters can be difficult to ascertain. *randomize_on_errors* is a means of automatically randomizing parameters based on the approximate errors in the parameters as given in Eq. (4-10), where *T* is the current temperature and K is as defined in Eq. (4-3).

$$\Delta p_i = Q \, Sign\big(Rand(-1,1)\big)\sqrt{0.02 \, T/(K \, A_{ii})} \qquad (4\text{-}10)$$

Q is a scaling factor determined such that convergence to a previous parameter configuration occurs 7.5% of the time on average. When *randomize_on_errors* is used, relative variation in *temperature*(s) are significant and not absolute values. The macro *Auto_T* includes *quick_refine*, *randomize_on_errors* and a *temperature* regime. It has shown to be adequate for a wide range of simulated annealing examples, see example CIME-Z-AUTO.INP.

4.17 ... Adaptive step size using randomize_on_errors

Use of *randomize_on_errors* result in an adaptive-step-size determination during simulated annealing; in many cases the complex temperature regime found in the macro *Auto_T* can be replaced with a single temperature. The example CIME-Z-AUTO.INP demonstrates *randomize_on_errors* by using a very incorrect starting temperature of 0.1; the program quickly modifies the temperature to a more appropriate value. Output lines such as:

```
Breaking - randomize on errors revisit
```

indicate that a parameter configuration has been revisited and the temperature will be internally adjusted. Note, with *randomize_on_errors*, relative temperature values are pertinent and not absolute values.

4.18 ... Criteria of fit

Trwp
   [*r_p* #][*r_wp* #][*r_exp* #][*gof* #][*r_p_dash* #][*r_wp_dash* #][*r_exp_dash* #]
   [*weighted_Durbin_Watson* #]

Global and *xdd* dependent refinement indicators. Keywords ending in "_dash" correspond to background subtracted values.

Table 4-2. Criteria of fit (Young , 1993). $Y_{o,m}$ and $Y_{c,m}$ are the observed and calculated data respectively at data point m; Bkg_m the background, M the number of data points, N the number of parameters, w_m the weighting given to data point m; for counting statistics $w_m = 1/\sigma(Y_{o,m})^2$ where $\sigma(Y_{o,m})$ is the error in $Y_{o,m}$. And $I_{o,k}$ and $I_{c,k}$ the observed and calculated intensities of the kth reflection. [1]) background corrected.

R-pattern, Rp'	$R_p = \dfrac{\sum |Y_{o,m} - Y_{c,m}|}{\sum Y_{o,m}}$	$R'_p = \dfrac{\sum |Y_{o,m} - Y_{c,m}|}{\sum |Y_{o,m} - Bkg_m|}$
R-weighted pattern, Rwp, [1]Rwp'	$R_{wp} = \sqrt{\dfrac{\sum w_m (Y_{o,m} - Y_{c,m})^2}{\sum w_m Y_{o,m}^2}}$	$R'_{wp} = \sqrt{\dfrac{\sum w_m (Y_{o,m} - Y_{c,m})^2}{\sum w_m (Y_{o,m} - Bkg_m)^2}}$
R-expected, [1]Rexp'	$R_{exp} = \sqrt{\dfrac{M - N}{\sum w_m Y_{o,m}^2}}$	$R'_{exp} = \sqrt{\dfrac{M - N}{\sum w_m (Y_{o,m} - Bkg_m)^2}}$
Goodness of fit, GOF	$GOF = chi = \dfrac{R_{wp}}{R_{exp}} = \sqrt{\dfrac{\sum w_m (Y_{o,m} - Y_{c,m})^2}{M - N}}$	
R-Bragg	$R_B = \dfrac{\sum |I_{o,k} - I_{c,k}|}{\sum I_{o,k}}$	

Durbin–Watson, d, 1971; Hill & Flack, 1987	$$d = \frac{\sum_{m=2}^{M}(\Delta Y_m - \Delta Y_{m-1})^2}{\sum_{m=1}^{M}(\Delta Y_m)^2} \;;\quad \Delta Y_m = Y_{o,m} - Y_{c,m}$$

5. ..PEAK GENERATION AND "PEAK_TYPE"

Convolution implies integration; a function analytically integrated is exact whereas numerical integration is an approximation with an accuracy dependent on the step size used for integration. Accurate numerical convolution is used when analytical convolution is not possible; this makes it possible to include complex functions in the generation of peak shapes. Laboratory instrument aberration functions mostly require numerical convolution. This process of convolution, from a fundamental-parameters perspective (Cheary & Coelho, 1992; Cheary *et al.*, 2004), is an approximation; second order effects and higher are typically neglected. These approximations are valid except for extreme cases that are unlikely to exist in practice, for example, axial divergence with Soller slits acceptance angles that are greater than about 12 degrees.

5.1..... Source emission profiles

Generation of the emission profile is the first step in peak generation. It comprises EM lines, EM_k, each of which is a Voigt comprising the parameters *la*, *lo*, *lh* and *lg*. The reserved parameter name *Lam* is assigned the *lo* value of the EM_k line with the largest *la* value, this EM_k will be called *EMREF*. It is used to calculate d-spacings. The interpretation of EM data is dependent on *peak_type*. For all peak types, the position $2\theta_k$ calculated for an emission line for a Bragg position of 2θ is determined as:

$$2\theta = ArcSin\left(\frac{EM(k, lo)}{2d}\right)\frac{360}{\pi} \qquad \text{where} \qquad 2d = \frac{EMREF(lo)}{Sin(\theta)}$$

2θ for *xo_ls* phases corresponds to the *xo* parameter. 2θ for *d_ls* phases is given by the Bragg equation $2\theta = ArcSin(Lam/(2\ d))\ 360/Pi$ where *d* corresponds to the value of the *d* parameter. 2θ values for *str* and *hkl_ls* phases are calculated from the lattice parameters. The $FWHW_k$ in °2θ for an EM_k line is determined from the relations provided in Table 5-1. When *no_th_dependence* is defined then the calculation of $2\theta_k$ is determined from

$$2\theta_k = 2\theta + EM(lo, i)$$

The macro *No_Th_Dependence* can be used when refining on non–X-ray data or fitting to negative 2θ values (see example NEGX.INP). The x-axis extent (x1, x2) to which an EM line is calculated is determined by:

$$\frac{Intensity\ of\ EM(i, x = x1 = x2)}{Intensity\ of\ EMREF(x = 0)} = ymin\_on\_ymax$$

The default for *ymin_on_ymax* is 0.001. Emission profile data have been taken from Hölzer *et al.* (1997) and are stored in *.LAM files in the LAM directory.

Table 5-1. $FWHW_k$ in °2θ for an EM_k line for the different peak types.

FP peak type	$FWHM_k = \left(\dfrac{EM(k, lh)}{LAM}\right)\dfrac{Tan(\theta)360}{\pi}$

PV peak type	$FWHM_k = \dfrac{\text{pv\_fwhm} \;\; EM(k, lh)}{EMREF(lh)}$
SPVII peak type	$FWHM_k = \dfrac{(h_1 + h_2)EM(k, lh)}{EMREF(lh)}$
SPV peak type	$FWHM_k = \dfrac{\left(spv_{h_1} + spv_{h_2}\right)EM(k, lh)}{EMREF(lh)}$

5.2..... Peak generation and peak types

Phase peaks P are generated as follows:

$$P = \text{Get}(scale)\,\text{Get(all\_scale\_pks)}\,\text{EM}(peak\_type) \otimes \text{Convolutions} \qquad (5\text{-}1)$$

where the emission profile (EM) is first generated with emission profile lines of type *peak_type*; the symbol $\otimes$ denotes convolution. Peaks are then convoluted with any defined convolutions, multiplied by the *scale* parameter, multiplied by any defined *scale_pks*, and then multiplied by an intensity parameter. For *xo_ls*, *d_ls* and *hkl_ls* phases the intensity is given by the I parameter. For *str* phases it corresponds to the square of the structure factor $F^2(hkl)$. Convolutions are normalized and do not change the area under a peak except for the *capillary_diameter_mm* and *lpsd_th2_angular_range_degrees* convolutions. The area under the emission profile is determined by the sum of the *la* parameters; typically, they add up to 1. The definitions of the pseudo-Voigt and PearsonVII functions are provided in Table 5-2.

Table 5-2. Unit area peak types. *x* corresponds to $(2\theta - 2\theta_k)$ where $2\theta_k$ is the position of the kth reflection. *fwhm* corresponds to the Full Width at Half Maximum. η is the PV mixing parameter. The '1' and '2' subscripts corresponds to the left and right of the split functions.

Gaussian	$G_{UA}(x) = \left(\dfrac{g_1}{fwhm}\right)Exp\left(\dfrac{-g_2 x^2}{fwhm^2}\right), \quad g_1 = 2\sqrt{\dfrac{Ln(2)}{\pi}}, \quad g_2 = 4Ln(2)$
Lorentzian	$L_{UA}(x) = \dfrac{\left(\dfrac{l_1}{fwhm}\right)}{\left(1 + \dfrac{l_2 x^2}{fwhm^2}\right)}, \quad l_1 = \dfrac{2}{\pi}, \quad l_2 = 4$
PseudoVoigt	$PV = \eta L_{UA}(x) + (1 - \eta)G_{UA}(x)$

Split PearsonVII SPVII	$SPVII = PVII_{Left} + PVII_{Right}$
	$PVII_{Left} = a^{-1}(1 + b_1 x^2)^{-m_1}, \quad \text{for } (-\infty < x < 0)$
	$PVII_{Right} = a^{-1}(1 + b_2 x^2)^{-m_2}, \quad \text{for } (0 < x < \infty)$
	$a = \frac{1}{2}\,\Gamma(\frac{1}{2})\left(\dfrac{\Gamma(m_1 - \frac{1}{2})}{\Gamma(m_1)\sqrt{b_1}} + \dfrac{\Gamma(m_2 - \frac{1}{2})}{\Gamma(m_2)\sqrt{b_2}}\right)$
	$b_1 = \left(2^{\frac{1}{m_1}} - 1\right)h_1^{-2}, \quad b_2 = \left(2^{\frac{1}{m_2}} - 1\right)h_2^{-2}$
	$fwhm_1 = 2\,h_1, \quad fwhm_2 = 2\,h_2, \quad fwhm = h_1 + h_2$
Split PseudoVoigt SPV	$SPV = \dfrac{2\left(PV_{Left} + a\,PV_{Right}\right)}{1 + a}$
	$PV_{Left} = PV(h_1, \eta_1), \quad \text{for } (-\infty < x < 0)$
	$PV_{Right} = PV(h_2, \eta_2), \quad \text{for } (0 < x < \infty)$
	$a = \dfrac{PV_{Left}(x = 0)}{PV_{Right}(x = 0)}$
	$fwhm_1 = 2\,h_1, \quad fwhm_2 = 2\,h_2, \quad fwhm = h_1 + h_2$

Lorentzian and Gaussian convolutions using *lor_fwhm* and *gauss_fwhm* equations are analytically convoluted with FP and PV peak types and numerically convoluted with the SPVII and SPV peak types. These numerical convolutions have a high degree of accuracy as they comprise analytical Lorentzian and Gaussian functions convoluted with straight line segments. For FP and PV peak types, the first defined *hat* convolution is convoluted analytically. Additional hat convolutions for all peak types are convoluted numerically. For classic analytical full pattern fitting the macros *PV_Peak_Type*, *PVII_Peak_Type*, *TCHZ_Peak_Type* can be used. These macros use the following relationships to describe profile width as a function of 2θ.

PV_Peak_Type	*PVII_Peak_Type*
fwhm = ha + hb tan(θ) + *hc*/cos(θ)	*fwhm₁ = fwhm₂ = ha + hb* tan(θ) + *hc*/cos(θ)
η = *lora + lorb* tan(θ) + *lorc*/cos(θ)	*m1 = m2* = 0.6 + *ma + mb* tan(θ) + *mc*/cos(θ)
where *ha, hb, hc, lora, lorb, lorc* are refineable parameters.	where *ha, hb, hc, ma, mb, mc* are refineable parameters.

TCHZ_Peak_Type: The modified Thompson-Cox-Hastings pseudo-Voigt "TCHZ" is defined as (e.g. Young, 1993, see example ALVO4_TCH.INP):

η = 1.36603 *q* - 0.47719 q^2 + 0.1116 q^3

where $q = \Gamma_L / \Gamma$

$\Gamma = (\Gamma_G^5 + A\Gamma_G^4\Gamma_L + B\Gamma_G^3\Gamma_L^2 + C\Gamma_G^2\Gamma_L^3 + D\Gamma_G\Gamma_L^4 + \Gamma_L^5)^{0.2}$ = *fwhm*

A = 2.69269, B = 2.42843, C = 4.47163, D = 0.07842

$\Gamma_G = (U \tan^2\theta + V \tan\theta + W + Z / \cos^2\theta)^{0.5}$

$$\Gamma_L = X \tan\theta + Y / \cos\theta$$

with U, V, W, X, Y, Z as refined parameters.

5.3..... Convolution and the peak generation stack

The emission profile of a peak P0 of a certain peak type (i.e. FP, PV etc...) is first calculated and placed onto a 'Peak calculation stack'. P0 analytically includes *lor_fwhm* and *gauss_fwhm* convolutions for FP and PV peak types and additionally one *hat* convolution if defined; the *hat* convolution is included analytically only if its corresponding *num_hats* has a value of 1 and if it does not take part in stack operations. Further defined convolutions are convoluted with the top member of the stack. The last convolution should leave the stack with one entry representing the final peak shape. The following keywords allow for manipulation of the Peak calculation stack:

    [*push_peak*]...
    [*bring_2nd_peak_to_top*]...
    [*bring_n_peak_to_top* !E]...
    [*add_pop_1st_2nd_peak*]...
    [*scale_top_peak* E]...
    [*set_top_peak_area* E]...

push_peak duplicates the top of the stack; *bring_2nd_peak_to_top* brings the second entry to the top of the stack, *bring_n_peak_to_top* brings the nth peak to the top (n=0 corresponds to the top of the stack) and *add_pop_1st_2nd_peak* adds the top entry to the second most recent entry and then pops the stack. *scale_top_peak* scales the peak at the top of the stack. As an example, consider the generation of back-to-back exponentials as required by GSAS time of flight peak shape 3:

```
push_peak
    prm a0   481.71904 del = 0.05 Val + 2;
    prm a1  -241.87060 del = 0.05 Val + 2;
    exp_conv_const = a0 + a1 / D_spacing;
bring_2nd_peak_to_top
    prm b0  -3.62905 del = 0.05 Val + 2;
    prm b1   6.44536 del = 0.05 Val + 2;
    exp_conv_const = b0 + b1 / D_spacing^4;
add_pop_1st_2nd_peak
```

The first statement *push_peak* pushes P0 onto the stack leaving two peaks on the stack:

    Stack = P0, P0

The top member is then convoluted by the first *exp_conv_const* convolution, or;

    Stack = P0, P0 $\otimes$ *exp_conv_const*

where $\otimes$ denotes convolution. *bring_2nd_peak_to_top* results in the following:

    Stack = P0 $\otimes$ *exp_conv_const*, P0

and the next convolution results in:

$$\text{Stack} = \text{P0} \otimes \textit{exp\_conv\_const}, \text{P0} \otimes \textit{exp\_conv\_const}$$

Thus, the stack contains two peaks convoluted with exponentials. The last statement *add_pop_1st_2nd_peak* produces:

$$\text{Stack} = \text{P0} \otimes \textit{exp\_conv\_const} + \text{P0} \otimes \textit{exp\_conv\_const}$$

Convolutions applied to peaks are normalized after convolution. Thus, the following, from WIF David's macro *wifd_mic_moderator*, will give unintended peak shapes:

```
push_peak                             ' first peak
   scale_top_peak = 1 - storage
bring_2nd_peak_to_top                 ' second peak
   exp_conv_const = -Ln(0.001) / (taus_0 + taus_1 / lam^2);
   scale_top_peak = storage;
add_pop_1st_2nd_peak
```

where the ratio of the areas of the first peak to the second peak won't be (1-storage)/storage. This can be remedied by normalizing the *exp_conv_const* aberration as follows:

```
push_peak
   scale_top_peak = 1 - storage;
bring_2nd_peak_to_top
   exp_conv_const = -Ln(0.001) / (taus_0 + taus_1 / lam^2);
   scale_top_peak = storage Yobs_dx_at(Xo);
add_pop_1st_2nd_peak
```

However, not all aberrations are easily normalized; *set_top_peak_area* overcomes this problem by normalizing the area itself in situ. The INP segment can now be written as:

```
push_peak
   set_top_peak_area = 1 - storage;
bring_2nd_peak_to_top
   exp_conv_const = -Ln(0.001) / (taus_0 + taus_1 / lam^2);
   set_top_peak_area = storage;
add_pop_1st_2nd_peak
```

5.4..... Speed / Accuracy and peak_buffer_step

For computational efficiency, phase peaks are calculated at predefined 2θ intervals in a "peaks buffer". In between peaks are determined by stretching and interpolating. Use of the peaks buffer dramatically reduces the number of peaks calculated. Typically, no more than 50 to 100 peaks are necessary to accurately describe peaks across a whole diffraction pattern. The following key-words affect the accuracy of phase peaks:

[*peak_buffer_step* !E]
[*convolution_step* #1]
[*ymin_on_ymax* #]
[*aberration_range_change_allowed* !E]

Default values for these are typically adequate. *peak_buffer_step* determines the maximum x-axis spacing between peaks in the peaks buffer, it has a default value of

500*_Peak_Calculation_Step_. A value of zero will force the calculation of a new peak in the peaks buffer for each peak of the phase. Note that peaks are not calculated for x-axis regions that are void of phase peaks. _convolution_step_ defines an integer corresponding to the number of calculated data points per measurement data point used to calculate the peaks in the peaks buffer, see _x_calculation_step_. Increasing the value for _convolution_step_ improves accuracy for data with large step sizes or for peaks that have less than 7 data points across the FWHM. _ymin_on_ymax_ determines the x-axis extents of a peak (see also section 5.1). _aberration_range_change_allowed_ describes the maximum allowed change in the x-axis extent of a convolution aberration before a new peak is calculated for the peaks buffer. For example, in the case of _axial_conv_ the spacing between peaks in the peaks buffer should be small at low angles and large at high angles. _aberration_range_change_allowed_ is a dependent of the peak type parameters and convolutions as shown in Table 5-3. Small values for _aberration_range_change_allowed_ reduces the spacing between peaks in the peaks buffer and subsequently increase the number of peaks in the peaks buffer.

Table 5-3. Default values for _aberration_range_change_allowed_.

Parameter	Default _aberration_range_change_allowed_
_m1_, _m2_	0.05
_pv_lor_, _spv_l1_, _spv_l2_	0.01
_h1_, _h2_, _pv_fwhm_, _spv_h1_, _spv_h2_	_Peak_Calculation_Step_
hat, _axial_conv_, _whole_hat_, _half_hat_	_Peak_Calculation_Step_
_one_on_x_conv_, _exp_conv_const_, _circles_conv_	_Peak_Calculation_Step_
_lor_fwhm_, _gauss_fwhm_	_Peak_Calculation_Step_

5.5..... The peaks buffer, speed and memory considerations

Anisotropic peak shapes result in the peaks buffer holding as many peaks as there are hkls. For problems with 100,000s of peaks the calculation time and storage of the peaks buffer can be prohibitive. This situation can be mitigated using the phase dependent _peak_buffer_based_on_:

> [_str_ | _hkl_ls_ | _xo_ls_ | _d_ls_]
> > [_peak_buffer_based_on_ !E [_peak_buffer_based_on_tol_ !E]]...

When _peak_buffer_based_on_ is defined, the usual means of determining the size of the peak buffer is over-ruled. Instead peaks are grouped according to the _peak_buffer_based_on_ criterion. For example, to insert a peak into the peak buffer at x-axis intervals of 1 then the following can be used:

```
peak_buffer_based_on = Xo; peak_buffer_based_on_tol 1
```

Thus, peaks with similar _Xo_'s, as defined by _peak_buffer_based_on_tol_, are grouped. Occasionally peaks that are a function of hkls have groups of hkls that are of the same peak shape and at a similar x-axis position. The following demonstrates how to group these peaks such that a single peak shape is calculated.

```
peak_buffer_based_on = Xo; peak_buffer_based_on_tol 0.01
peak_buffer_based_on = sh; peak_buffer_based_on_tol 1e-7
```

where *sh* can be a spherical harmonics parameter or an equation describing hkl dependence or a *march_dollase* parameter. When more than one *peak_buffer_based_on* is defined then peak groups obey all of the *peak_buffer_based_on*'s. *peak_buffer_based_on* disables the peak stretching procedures and any defined *aberration_range_change_allowed*. *peak_buffer_based_on* can be a function of the reserved parameters *H*, *K*, *L*, *M*, *D_spacing*, *X*, *Xo*, *Th*.

Depending on the problem, smaller values such as 1e-7 can significantly reduce the number of peaks stored in the peaks buffer (a factor of 15 at times) without significantly affecting Rwp. A negative value for *peak_buffer_based_on_tol* will force a calculation for each peak resulting in independent hkl peak shapes, for example:

```
peak_buffer_based_on 1 peak_buffer_based_on_tol -1
```

5.6..... An Accurate Voigt

[*more_accurate_Voigt*] can be used to override the default Pseudo-Voigt approximation to the Voigt. It decreases the error (Voigt_approx – Voigt_true) by a factor of ~100. Defining G as the FWHM of a Gaussian and L as the FWHM of a Lorentzian; the screen shots below show fits to a range of G convoluted with L, resulting in Voigts, with L varying from 0.01 to 0.09 and G+L=1. Fitting to the Voigts using pseudo-Voigts we get
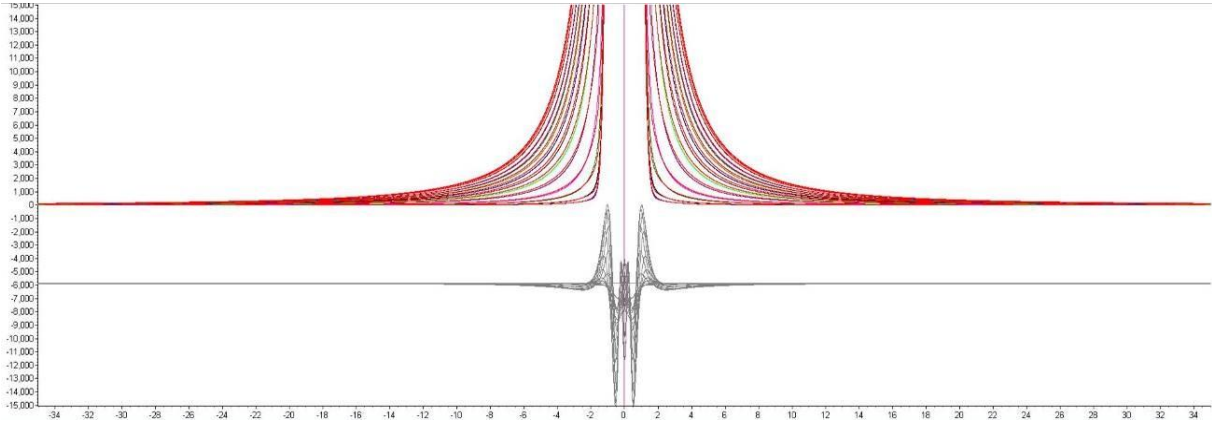
Fitting to the Voigts using the accurate calibration results in the small difference plots seen in the following:
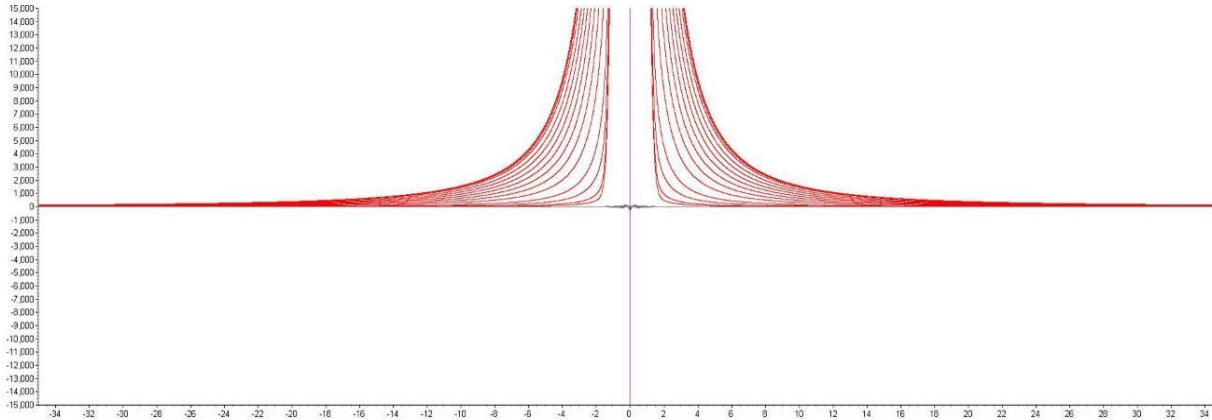
Rescaling the plot vertically to show the whole scan gives
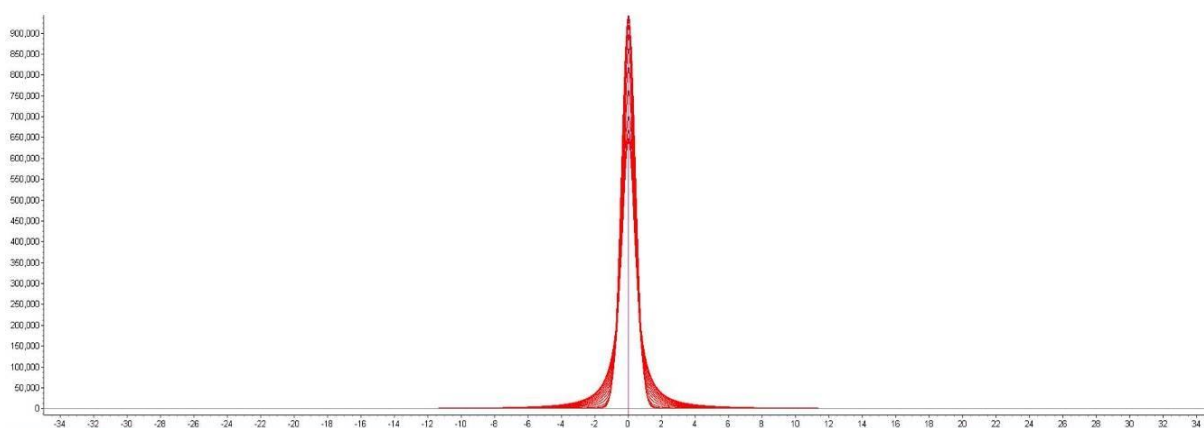
Note that the difference plot simply appears as a straight line. The *more_accurate_Voigt* calibration is accurate and fast. It fits to each true Voigt the following:

```
fit_obj = a1 (2 Sqrt(Ln(2) / Pi) / f1) Exp(-4 Ln(2)(X / f1)^2);
fit_obj = a2 (2 Sqrt(Ln(2) / Pi) / f2) Exp(-4 Ln(2)(X / f2)^2);
fit_obj = a3 (2 / (Pi f3)) / (1 + 4 (X / f3)^2);
fit_obj = a4 (4 / (Pi f4)) / (1 + 4 (X / f4)^2)^2;
```

One thousand sets of a0, a1, a2, a3, f0, f1, f2, f3 parameters were determined by fitting to 1000 true Voigts with L varying from 0 to 1 in steps of 0.001.

numerical_lor_gauss_conv creates a 'true' Voigt by numerically convoluting Gaussians with Lorentzians; the extents to which these aberrations are calculated can be defined using *numerical_lor_ymin_on_ymax* (default of 0.0001). The CREATE.INP file in the TEST_EXAMPLES\VOIGT-APPROX directory uses *numerical_lor_gauss_conv* where the amount of Lorentzian is entered as a number out of a 1000. A value of 500 would yield a Voigt with a Lorentzian FWHM of 0.5 and a Gaussian FWHM of 0.5. The generated true Voigt is calculated by numerically convoluting a *lor_fwhm* with a *gauss_fwhm*. The generated true Voigt is saved to a file with the name VOIGTNNNN.XY, where NNNN corresponds to a number between 0 and 1000. The file generated contains 100,000 data points. The step size used in the convolutions is as small as 0.0005 when using a *convolution_step* of 4.

TOPAS uses an FFT to perform the double summation of the convolution. However, for lor > 500, the convolution itself comprises an analytical Lorentzian with a Gaussian comprising straight line segments. For lor < 500 then an analytical Gaussian is convoluted with a Lorentzian comprising straight line segments.

- The file FIT-PV.INP fits a pseudo-Voigt to the generated true Voigt.
- The file FIT-MORE.INP fits to the generated true Voigt using equivalent *fit_obj*'s.
- The file FIT-OBJ.INP fits *fit_obj*'s to the generated true Voigt.

The difference plot from FIT-PV.INP is in the order of 500 to 1000 times larger than the difference plot from FIT-MORE.INP.

6. ..FAST SIMULTANEOUS REFINEMENT OF 1000S OF PATTERNS

[*str*...]
    [*peak_buffer_similar_tag* !E]
    [*hkl_similar_tag* !E]
	<u>Example</u>
	TEST_EXAMPLES\1000S-OF-PATERNS\FIT.INP

Previous version of TOPAS applied threading at the phase level; Version 7 extends threading to the *xdd* level. Computers with many processors now show large improvements in speed when many diffraction patterns are refined simultaneously. Refining on 1000s of patterns with many threads is also memory intensive. To reduce memory usage, TOPAS looks for items that are unique and stores only one. For example, operating on an INP file with 1000s of data files and 1000s of phases, the program:

- Calculates and stores only unique structures.
- Calculates and stores only unique sets of hkls amongst the unique structures.
- Calculates and stores only unique *bkg* derivatives.
- Calculates and stores only unique peak positions and d-spacing amongst unique sets of hkls and lattice parameters.
- Calculates and stores only unique *scale_pks* equations.
- Calculates and stores only unique *th2_offset* equations.
- Calculates and stores only unique *pk_xo* equations.
- Calculates and stores only unique x-axis data (ie. *xdd* data files with the same x-axis).

For example, if 10000 data files were loaded each with 10000 identical sets of hkls, then only one set of hkls are stored saving approximately 1.6 Gbytes of memory. Operating on items that are unique also reduces calculations, for example, *scale_pks* equations that have identical (or similar) hkls are only calculated once; and similarly, for *th2_offset*, *pk_xo*, internal peak positions and internal d-spacings.

For phases with similar sets of hkls, then peaks-buffers can be calculated once using the *peak_buffer_similar_tag*. For example, only three peaks-buffers are calculated and stored in the following even through 8 phases are defined:

```
xdd …
    str … peak_buffer_similar_tag 1
    str … peak_buffer_similar_tag 2
    str … peak_buffer_similar_tag 3
    str … peak_buffer_similar_tag 1
xdd …
    str … peak_buffer_similar_tag 3
    str … peak_buffer_similar_tag 1
    str … peak_buffer_similar_tag 2
    str … peak_buffer_similar_tag 3
```

An exception is also thrown if peaks-buffers with similar *peak_buffer_similar_tag* are not actually similar; for example, the following will throw an exception as the CS_L parameters are different (ie. two independent parameters a1 and a2).

```
xdd …
    str … peak_buffer_similar_tag 1 CS_L(a1, 100)
xdd …
    str … peak_buffer_similar_tag 1 CS_L(a2, 100)
```

In some cases, sets of hkls are similar but not identical due to slightly different lattice parameters. In such cases *hkl_similar_tag* can be used to force the use of a single set of hkls resulting in reduced memory usage and improved speed. In general:

- Phases with the same peaks-buffer can have different *fit_obj*(s), *bkg*(s), *th2_offset*(s), *scale_phase_X*(s), *scale_pks*(s) or *pk_xo*(s).

- Phase with the same structural parameters (same sites) can have different *fit_obj*(s), *bkg*(s), *th2_offset*(s), *scale_phase_X*(s), *scale_pks*(s), *pk_xo*(s) and lattice parameters.

6.1..... Example refinement of 1000s of patterns

First, it may be best to set the number of threads in the file MAXNUMTHREADS.TXT to the number of physical CPU cores in your computer. FIT.INP creates a test pattern with 4001 data points when "*#define* CREATE__" is used. When "*#define* CREATE__" is commented out then refinement proceeds on many patterns, the number of which is stipulated by the *Num_Files_2* macro. Each pattern has:

- three structures with the number of hkls generated being 15, 33 and 52
- five unique *bkg* parameters
- four unique lattice parameters
- three unique scale parameters
- one unique zero error parameter
- one unique specimen displacement parameter
- one unique *LP_Factor* parameter

Global to all patterns are three CS_L and three CS_G parameters. This results in 12009 independent parameters for 1000 patterns, or 60009 independent parameters for 5000 patterns. The **A** matrix is sparse as seen in the (right) for 10 patterns.
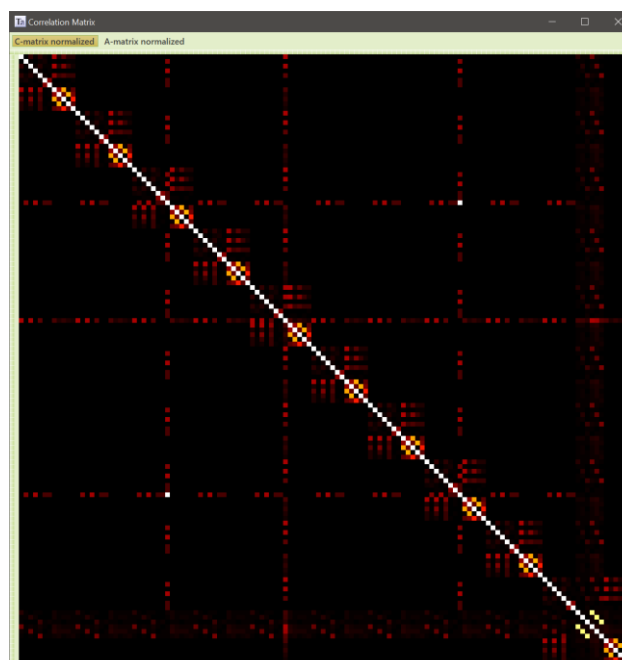
The following plots the time taken to perform 30 iterations as a function of Number-of-patterns for Version 6 and Version 7 with the use of *peak_buffer_similar_tag*; and additionally, for Version 7 without the use of *peak_buffer_similar_tag*.
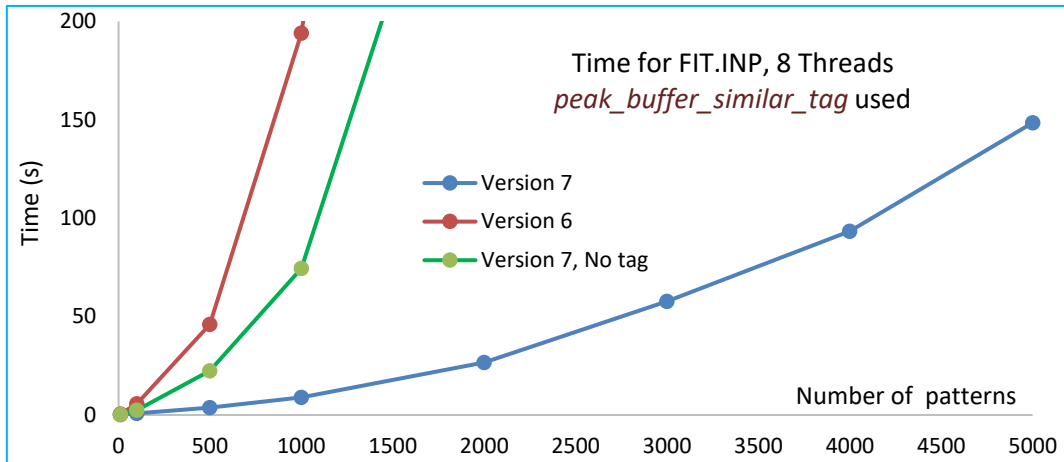
Seen is the very large reduction in refinement times when *peak_buffer_similar_tag* is used. Version 6 is considerably slower due to threading at the *str* level rather than the *xdd* level; Version 7 with *peak_buffer_similar_tag* is in fact well-over 120 times faster than Version 6. For Version 7 only, the following shows that the speed gain of Time with *peak_buffer_similar_tag* divided by Time without *peak_buffer_similar_tag* increases with Number-of-patterns.
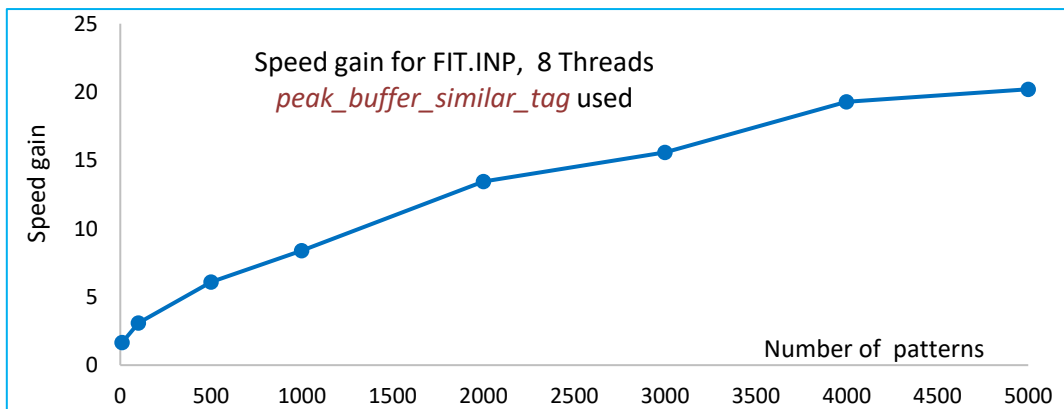
The following shows the reduction in memory usage for Version 7 compared to Version 6.
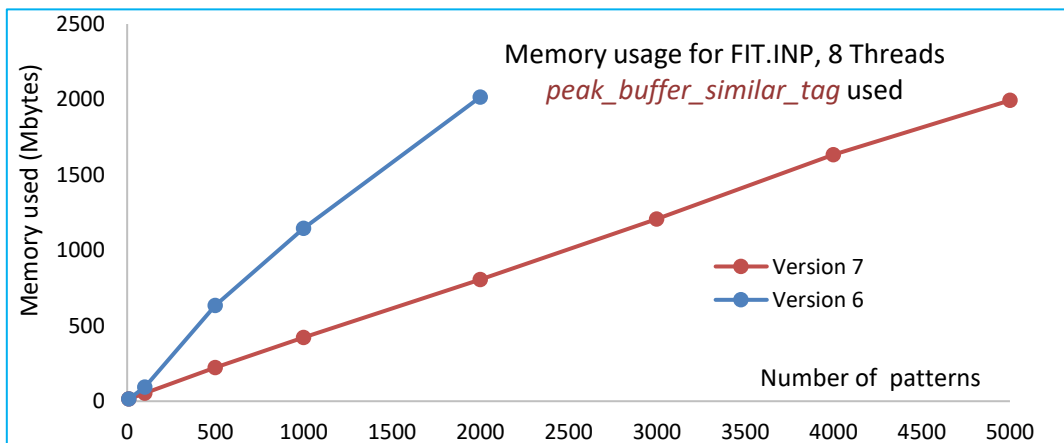
The above refinements for FIT.INP had three unique structures across all patterns. If the *beq* parameters of the three structures were refined independently then there would be 3*Number-of-

patterns structures and the times for Version 7 with *peak_buffer_similar_tag* increase by approximately 50%. This increase is modest considering the increase in Numbed-of-parameters as well as the increase in structure factor calculations.

7. ..AMAZON EC2 CLOUD COMPUTING

A cloud version of TOPAS can be run on multiple virtual computers on the Amazon Web Services (AWS) cloud platform. The process is seamlessly driven from the GUI version of TOPAS/TOPAS-Academic where launching an INP file on the cloud is a few mouse-clicks away. This gives users access to large computing resources where 1000s of virtual machines (VMs) can be utilized in a relatively inexpensive manner. Large simulated annealing problems taking weeks on a laptop can now be done in minutes. The process typically involves working interactively with TOPAS in Launch mode and performing initial preliminary refinements. Once the user is satisfied, the cloud version of the kernel, which we will call TC-Cloud, can be launched. Cloud operation is often performed in an interactive manner due to the speed of analysis; many Cloud runs need only last for 10 to 20 minutes depending on the number of VMs used.

The user does not install TC-Cloud; instead TC-Cloud is pre-installed on a Virtual Machine image called an Amazon Machine Image (AMI). The AMI for TC-Cloud is called TC-AMI. TC-AMI can be used to create many virtual machines each corresponding to a virtual Linux computer; we will call these TC-VMs. Each TC-VM can run multiple instances of TC-Cloud. To summarize:

- TA.EXE is the GUI version of TOPAS running on a local computer.
- TC-Cloud is the cloud version of TOPAS running on a VM.
- TC-AMI is an image of a VM with TC-Cloud installed.
- TC-VM is a VM created from TC-AMI.
- Many TC-VMs (500 for example) can be created/deleted at once.

The user is given a choice of VM type when launching TC-AMI to create TC-VMs. A large TC-VM can run more than one instance of TC-Cloud.

7.1..... Operation

TC-Cloud operates in a similar but not identical manner to TC.EXE. Importantly INP files are pre-processed before launching on the cloud; this ensures the use of local files such as TOPAS.INC and other *#include* files. Since the local TOPAS.INC is used then local emission profiles are used. Data files referenced in the INP file must reside in the same local directory as the INP file. This is normal practise and INP files should therefore not contain file paths. For example,

- this is valid on the cloud: *xdd* data.xy
- this is not valid on the cloud: *xdd* data\data.xy

File names on Linux are case sensitive. It is therefore important to use the correct case when referring to file names within INP files. The following keywords can be included in INP files but have been disabled:

append_bond_lengths	*do_errors_include_penalties*	*out_prm_vals_per_iteration*
atom_out	*do_errors_include_restraints*	*phase_out*
A_matrix	*index*	*phase_out_X*
A_matrix_normalized	*num_runs*	*process_times*
bootstrap_errors	*out*	*system_after_save_OUT*
C_matrix	*out_file*	*system_before_save_OUT*

C_matrix_normalized	*out_prm_vals_dependents_filter*	*verbose*
do_errors	*out_prm_vals_filter*	*view_structure*
	out_prm_vals_on_convergence	*xdd_out*

Many of these output data and as such are better left to the local computer.

7.2..... Pre-requisites

Signing up with Amazon AWS is required, see https://aws.amazon.com/. Also, necessary is TOPAS/TOPAS-Academic and a local computer to run TOPAS. TC-AMI comes with TOPAS/Academic Version 7; access to TC-AMI can be obtained from Alan Coelho. TC-VMs are monitored and terminated depending on user defined conditions. For example, VMs can be terminated when the best goodness of fit parameter (GOF) from all TC-VMs drop below a user defined value. This reduces running times for the TC-VMs and consequently running costs. The following points are important:

- Signing up with AWS does not incur a fee.
- Using non-free AWS resources do incur AWS fees.
- The user is responsible for all AWS costs.
- AWS fees can be reduced by reducing the use of AWS services.
- VMs created as spot instances are often 60 to 70% cheaper.
- Services can be reduced by:
  - Turning off unused VMs.
  - Deleting unused VMs.

7.3..... Pricing of AWS cloud resources

The following approximate pricing information are dependent on AWS and could change. Running TOPAS on AWS requires the use of VMs. Each VM in turn uses an EBS volume (a storage device). Use of both the VM and the EBS incur AWS fees, see:

For VMs: https://aws.amazon.com/ec2/pricing/on-demand/

For EBS volumes: https://aws.amazon.com/ebs/pricing/

Limited usage of a single core VM on Amazon AWS are free of charge for a period of one year. Large VMs (ones with many cores) are not free and charges are dependent on time usage. Pricing is on a per second basis for Linux VMs; the twin core VM *c5.large* is recommended for routine TC-Cloud usage; for the same core count, it is equivalent to an average high end laptop in computational speed and is priced at approximately ~0.034 cents USD (for spot instances) per hour. One hundred of these running for one hour will cost approximately $3.40 USD. Large saving, often up to 70%, can be realized by requesting *spot-instances*, see https://aws.amazon.com/ec2/spot/pricing/. The author has had no trouble getting regular access to 500 spot instances.

Each TV-VM is a Linux VM; it comes with an 8 Gbyte EBS volume which stores TC-Cloud and the operating system. EBS volumes are relatively inexpensive at 0.125 USD per Gbyte per month, or

$1 USD per month for each TC-VM. For one hundred VMs this small charge becomes $100 USD per month. It is therefore recommended that VMs are deleted after use to reduce costs. Creating and starting VMs takes one to two minutes.

Cloud storage is required in addition to VMs and associated EBS volumes. This storage is used to transfer data from the local computer to the VMs and visa-versa. AWS S3 cloud storage is used; its inexpensive at approximately $0.02 per Gbyte per month, see https://aws.ama-zon.com/s3/pricing/. File manipulation of S3 storage is provided. Running TC-Cloud typically requires a fraction of a Gbyte in S3 storage and hence common storage costs are negligible.

7.4..... AWS dashboard and operating TC-Cloud

AWS includes a comprehensive browser dashboard called EC2 Dashboard https://ap-southeast-2.console.aws.amazon.com/ec2/. In the case of running TC-Cloud, the dashboard is primarily used to create TC-VMs from TC-AMI as well as deleting files created on the S3 cloud storage. The rest of TC-Cloud operations are performed from TA.EXE. The important parts of EC2 Dashboard are circled in the following:
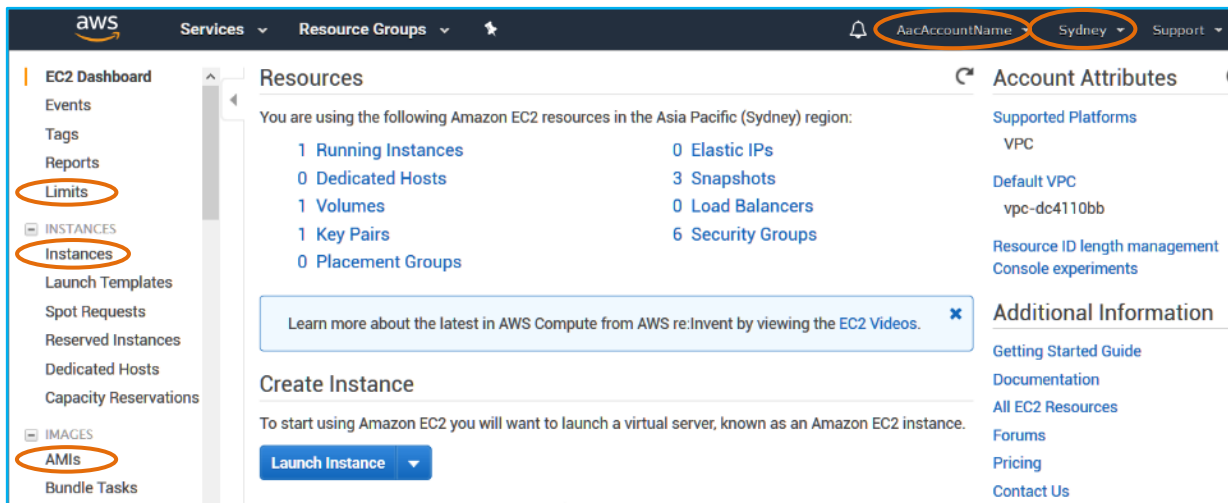
Note: AWS web screens may change due to improvements etc...; the general operation however should remain the same. Clicking on the *Account* (circled on the top) brings up account options which includes real time billing information (AWS cloud costs). Also, on the top is the AWS region being operated on. AWS operates on a regional basis; regions chosen should be in close geographical proximity to the local computer. This reduces response times and data transfer costs. TC-VMs are created by clicking on *AMIs*. Once created, details of TC-VMs for the selected region can be viewed by clicking on *Instances*. AWS limits the number of VMs available to 20 on most VM types; request for increasing this number can be made from the circled 'Limits' item. The author had no trouble getting regular access to 500 spot instance VMs.

7.5..... Installing AWS CLI on the local computer

For communicating with the TC- VMs; the local computer requires the installation of AWS Command Line Interface (CLI). The CLI can be trivially installed and downloaded from:

https://docs.aws.amazon.com/cli/latest/userguide/install-windows.html.

7.6 ..... Operating TC-Cloud from TOPAS (GUI)

After the preliminary setting up and testing of an INP file with TA.EXE on the local computer, the INP file can be fed to AWS for parallel operation on many VMs. Summing up the process we have:

1) Set up INP file and ensure it runs as expected on TA.EXE on the local computer.

2) Create a small number of VMs (3 for example) and ensure that the INP file runs as expected on the VMs.

3) Create many more VMs (user determined) and run the INP file on the VMs.

Stage-1 is normal TOPAS operation. Stage-2 involves creating a *job* (*.CLD files) from the 'Setup Cloud' tab in the GUI. Before creating a job its best to create a template that can be used for all jobs in the AWS region. Enter your *'Key pair file'*, the AWS *Region* being used and your *S3* bucket name details in the *Setup Cloud* tab; it should look something like:
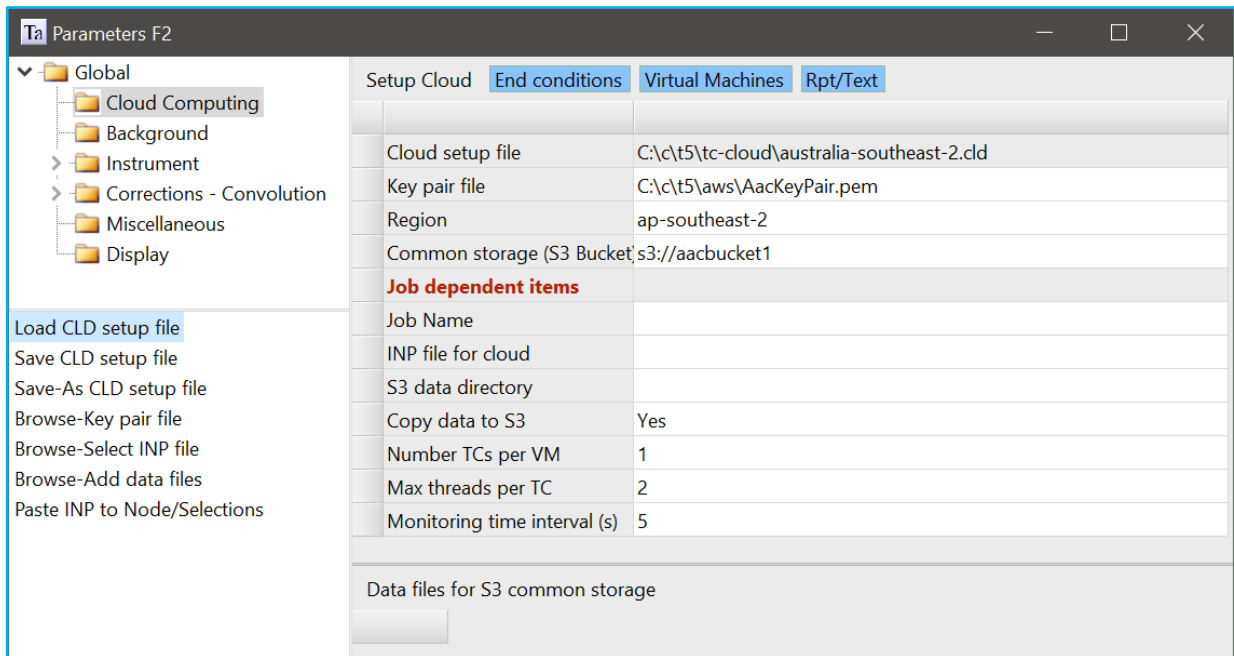
Save the details using '*Save-As CLD setup file*' to a file. Load this file when creating other CLD files. To run a job then enter the rest of the setup details; an example is:
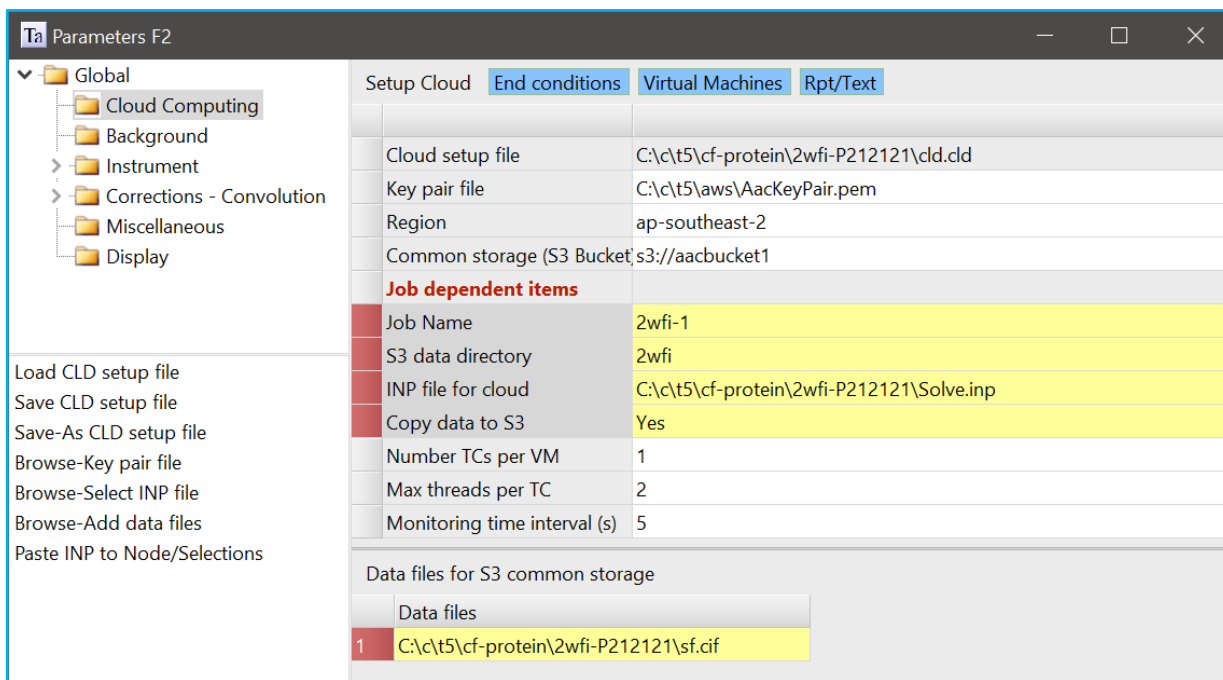
The highlighted lines require input to create a job. This input comprises the INP file to be run on the Cloud as well as the necessary data files. In the above example the INP file is placed in the S3 job directory called 2wfi-1 and the data file is placed in the S3 directory called 2wfi. S3 will therefore contain the following two directories:

s3://aacbucket1/swfi-1

s3://aacbucket1/swfi

The INP file as well as other communication files are copied to the job directory, 2wfi-1 in this case. The name of the INP file on S3 is changed to in.inp; in.inp is used in the retrieval of out from the VMs; it is unchanged during Cloud operation and it can be also viewed as a backup for the job. Each run on the Cloud requires a unique job name; an exception is thrown otherwise. Many jobs however, can use the same S3 data directory. In cases where many jobs are run sequentially, each using the same data files, then the '*Copy data to S3*' option can be set to No after the first job; this speeds up processing as copying large data files over the internet can be slow. CLD files contain information necessary for launching the INP file on the cloud. Once the information is entered, it becomes possible to view the created VMs in the 'Virtual Machines' tab, or:
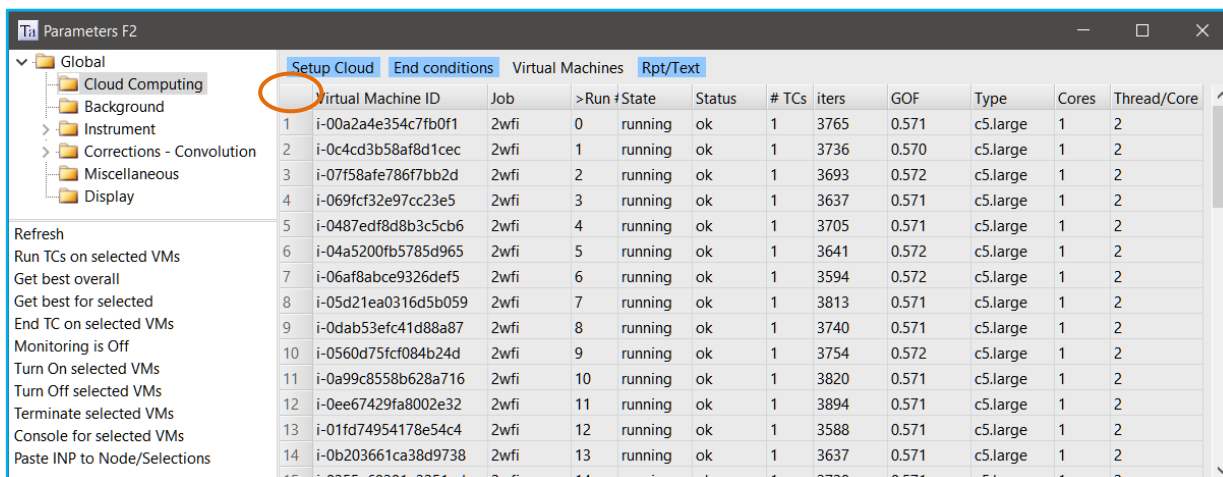
Data can be displayed in sorted order by double clicking on column headings. To launch the INP file on a VM then select the VM and click *'Run TC on selected VMs'*. To select all VMs then click on the empty rectangle circled. Only VMs with an *ok* Status can be launched. If a selected VM is *starting* or *pending* then *Status* will not be *ok*. The number of TCs running on each VM (typically one) is shown in the *# TCs* column. This data as well as other VM details maybe out-of-date; to show the latest then click on the *Refresh* option. The *iters* column shows the total number of refinement iterations executed on the respective VM; this number supplies a means of determining if a VM is running in an expected manner. For example, if *iters* has stopped increasing in an expected manner and *#TCs* is not zero then the running TCs have stopped operating in an expected manner.

Due to the speed of analysis, Cloud operation is often performed interactively. Running many jobs to investigate a problem, each taking 10 to 20 minutes and comprising 500 VMs, is common. Each job creates a directory on S3 which can be deleted after use using the AWS S3 dash-board; it looks like:
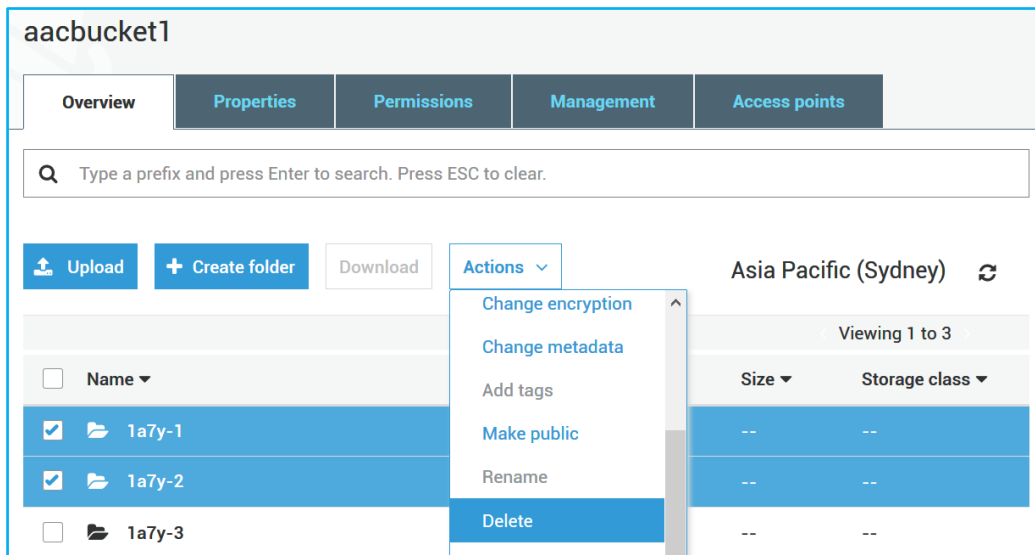
7.7 ..... Terminating/Stopping TC-VMs and tc-mon.a

Terminating or stopping TC-VMs reduces AWS fees. TC-VMs can be automatically stopped or terminated depending on *'End conditions'*, or:
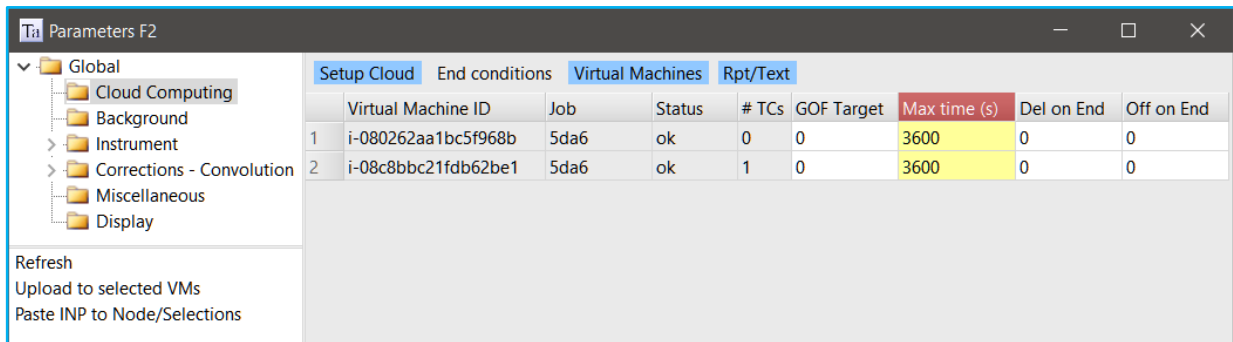
These conditions are uploaded to the VMs when a job is launched. On launching a job, a small monitoring program, called *tc-mon.a*, is started on each VM. This monitoring program reads the End conditions and monitors the running TCs. VMs are in turn terminated/stopped depending on the End conditions. From the local machine, the end conditions can also be uploaded after a job has

started using the *'Upload to selected VMs'* option. This option has no effect on VMs with a *Status* that is not *ok*. The *'Refresh'* option displays values as found on common storage for the job indicated in *'Setup cloud'* tab.

TCs running on VMs are terminated when the number or *iters*, as defined in the INP file, has been reached, or, when the CPU time allocated *'Max time (s)'* has been reached or when the overall best GOF falls below *'GOF Target'*. When there are no TCs running on a VM then the VM is stopped if *'Off on End'*=1; subsequently if *'Del on end'*=1 then the VM itself is terminated (deleted). Parameters for a typical job left unattended would be:

> Max time (s) = 10 60 60 = 10 hrs of running
>
> GOF Target = 10,   Off_on_End = 1,   Del_on_end = 1

For interactive use, the user can manually terminate TCs and VMs; the termination parameters could therefore look something like:

> Max time (s) = 0
>
> GOF_Target = 10, Off_on_End = 0, Del_on_end = 0

A *'Max time (s)'* of zero (the default) disables the ending of TCs on a time basis. *'Max time (s)'* on VMs can be entered as an equation by starting the equation with an equal sign. For example, '= 24 60 60' could be used to enter 24hrs.

7.8..... Powering off TC-VMs after 100 minutes of inactivity

In addition to the terminating/stopping criteria of section 7.7, VMs are automatically powered off (stopped but not terminated) after 100 minutes of TC-Cloud inactivity including inactivity on VM start-up. The net effect is that VMs are stopped after 100 minutes of TC-cloud not being run. Situations where 100 minutes of inactivity is possible include internet-down situations as well as users forgetting to power-off or terminate VMs. For example, the fee incurred for forgetting to turn off 100 spot instance VMs would be ~3.40 USD. Typical usage comprises turning on 500 VMs and running many jobs interactively to analyse data. The VMs are not turned off during the running of these jobs.

7.9..... Retrieving the INP or FC file that gave the best GOF

Output from a job, corresponding to the best INP for Rietveld refinement, or, the best structure factors for charge-flipping, is stored on the S3 job directory. This storage to S3 from a job is independent of the local computer. The *'Get best overall'* downloads the output to a local directory from where INP file originated. The name given to the output is *Job-Name*.INP for Rietveld refinement or *Job-Name*.FC for charge-flipping. For example, for a job named 'PbSO4-1' and an input file with a path of C:\DATA\PBSO4.INP we get:

> 'INP File for cloud' = C:\DATA\PBSO4.INP
>
> *'Get best overall'* places output in C:\DATA\PBSO4 -1.INP

Once retrieved, the best INP file can be run on the local computer; in other words, the best fit from the cloud can be visually inspected with a few mouse clicks. If the VMs are available and not

stopped or *terminated*, then output from the individual VMs can be retrieved using the *'Get best for selected'* option; output is placed in the local computer in an identical to that described for *'Get best overall'*. Typical interactive operation therefore comprise viewing and partially running intermediate cloud results and making decisions based on those results.

7.10 ... Monitoring, TC-Cloud is independent of the local computer

The running of VMs can be monitored by the local computer using the *'Monitoring is On/Off'* option. When On, the best overall GOF is displayed in the text output of the '*Fit Dialog'* window at time intervals as defined in *'Monitoring time interval'* option of *'Setup cloud'* tab. Whilst jobs are running, the local computer can be used to run refinements independent of any running jobs. Jobs can be started on a laptop, left running overnight and the results viewed the next day.

7.11 ... Random number generator automatically seeded

The random number generator for both TC-Cloud (and TC.EXE on the local computer) is seeded such that the sequence of random numbers generated for any run is unique. Identical sequences can be generated by using the *seed* keyword with an integer (corresponding to a seed number) placed after it.

7.12 ... CLOUD__ #define and Get(cloud_run_number)

The pre-processor directive of '*#define* CLOUD__' is automatically included at the start of INP files running on VMs. This allows blocks of INP script to be conditionally included/excluded from cloud runs making it easy to run the same INP file in both the cloud and on the local computer. For example, the following is useful in the case of charge-flipping:

```
charge_flipping
    #ifdef CLOUD__
        randomize_initial_phases_by = Rand(-180, 180);
    #else
        set_initial_phases_to job-name.fc
    #endif
```

Here the state of the best FC file found on the VMs can be determined by first executing the *'Get best overall'* option and then locally running the INP file. Also, available is Get(cloud_run_number) which returns the run number assigned to the corresponding VM with counting starting at 0. Get(cloud_run_number) returns -1 when running on the local computer. Example usage in terms of stacking faults could be:

```
macro & pa { Get(cloud_run_number+1)/102 }
generate_stack_sequences {
    number_of_sequences 200
    number_of_stacks_per_sequence 200
    Transition(1, lpc)
        to 1 = pa;        a_add = 2/3;    b_add = 1/3;
        to 2 = 1-pa;      a_add = 0;      b_add = 0;
    Transition(2, lpc)
        to 1 = 1-pa;      a_add = 0;      b_add = 0;
        to 2 = pa;        a_add = -2/3;   b_add = -1/3;
```

```
        }
```

7.13 ... 'Setup Cloud' details

Cloud setup file

Name of file containing cloud details for a job.

Key pair file

Name of file containing encrypted login information, see:

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html

This file needs to be read/write protected so that only one user can access; use Windows Explorer and Right-Click on the file to change its properties.

Region

Geographical region where VMs reside.

S3 Bucket

The name of the bucket for transferring data to and from the TC-VMs. Buckets are created and manipulated at https://s3.console.aws.amazon.com/s3/. By default, s3 buckets are private to the user. Once a bucket is created, directories within the bucket corresponding to the job name are automatically created on launching the TC-VMs. For example, for a job named *job-1* and a bucket called *my-bucket* then the following directory on s3 is created:

        s3://my-bucket/job-1

my-bucket are used for many jobs. Information stored on common storage are not deleted by TA.EXE running on the local computer; the user is therefore responsible for cleaning up unwanted files using the AWS S3 dash-board.

Job Name

Name of job. Job names cannot contain spaces.

S3 data directory

S3 directory where data files are stored for a job. More than one job can use a S3 data directory.

INP file for cloud

Input file to run on the cloud. The INP file can make use of the predefined pre-processor directive called CLOUD__. It can also make use of Get(cloud_run_number).

Number TCs per VM

Typically set to 1. The number of TC-Cloud instances to run on each TC-VM. The number of TCs per VM should not exceed the number of Cores as seen in *Cores* column of the *Virtual*

Machines tab. For example, the VM type of *c5.18xlarge* has 36 Cores each with 2 threads (intel hyper threading). The number of TCs therefore should not exceed 36. Information on EC2 instance types can be found at https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-optimize-cpu.html.

<u>*Max threads per TC*</u>

Typically set to 2 for *c5.large* VMs. The maximum number threads each TC can use. If zero, then each VM will be allowed to use the maximum number or threads. For VMs with more than one TC running then the maximum number threads should be set to:

$$Max\_threads\_per\_TC = (Virtual\ Cores) / Number\_TCs\_per\_VM$$

<u>*Monitoring time interval (s)*</u>

The time interval used when *'Monitoring is On'*.

7.14 ... 'Virtual Machines' tab options

<u>*Refresh*</u>

Refreshes VMs details corresponding to the region defined in the *'Setup cloud'* tab.

<u>*Run TC on selected VMs*</u>

Launches TC-Cloud on selected VMs.

<u>*Get best overall*</u>

Gets and processes the best output from common storage for the job defined in *Setup cloud* and places the result in the directory where the original INP file came from. For Rietveld refinement the retrieved output is placed in a file called *job-name*.INP. For charge-clipping, the retrieved output (structure factors) is placed in a file called *job-name*.FC. Files placed in common storage persists and are therefore available even after the job's VMs are deleted.

<u>*Get best for selected*</u>

Gets and processes the best output from a selected VM and places the result in the directory of the original INP file. The selected VM must be *On*. For Rietveld refinement the retrieved output is placed in a file called *job-name*.INP. For charge-clipping, the retrieved output (structure factors) is placed in a file called *job-name*.FC.

<u>*End TC on selected VMs*</u>

Stops any TC-Clouds running on selected VMs. On termination of the TCs, the VMs are turned off if their corresponding Off_on_End=1; in turn VMs are terminated if their corresponding Del_on_End=1.

<u>*Monitoring is On/Off*</u>

Starts/Stops monitoring. When monitoring is On, the best GOF as found by the TC-VMs for the job defined in *'Setup cloud'* is displayed in the Fit Dialog.

<u>*Turn On selected VMs*</u>

Turns selected VMs On.

<u>*Turn Off selected VMs*</u>

Turns selected VMs Off.

<u>*Console for selected VMs*</u>

Log-in to the selected VMs creating terminal windows for each. Can be useful for trouble shooting.

7.15 … Creating TC-VMs – Spot Instances

TC-VMs are created from the EC2 dashboard. To create 200 VMs, for example, click on the *AMIs* option and then click on the *TC-AMI-n* AMI. *n* corresponds to the latest TC-AMI version. Then click on *Launch* to bring up *'Choose an Instance Type'* screen. Choose an appropriate VM type; for refinements that require less than 4Gbytes of memory then choose *c5.large*. The amount of memory required for each TC can be determined by first running the INP file on the local machine and viewing the Windows Task Manager. Once the VM type is chosen, proceed to the next screen *'Configure Instance Details'*:

Set *'Number of instances'* to 200 and set the *'IAM role'* to *'ecsInstanceRole'*. Select *'Request Spot instances'*. Spot instances are often 60 to 70% cheaper; the user is informed when spot instances

are unavailable; the author has had no difficulty obtaining 500 spot instances on a regular basis. Proceed to the *'Configure Security Group' screen'* and set the *Source* to *'My IP'*; ie.

Click on *'Review and Launch'* to Launch the creation of the TC-VMs. Creation should take one to two minutes. Use the TA *Refresh* option of *'Virtual Machines'* to see the status of VMs; VMs with a *Status* of *ok* are ready to run. Once all the VMs are created, the *'Run TCs on selected VMs'* option from the *Virtual Machines* tab can be used to launch the job on the selected VMs.

7.16 ... Choosing the optimum VM type

The most appropriate VM for TOPAS type problems are *c5.large* where memory usage is less than 4 Gbytes. However, a problem that uses 20 Gbytes of memory would need a larger VM. The problem could be a large charge flipping problem or indeed a large Rietveld refinement simulated annealing problem with 1000s of parameters. Memory usage prior to launching on the Cloud can be determined using the local computer. The VM type chosen should therefore be one than has more memory and the maximum memory usage seen on the local computer. Only c* types (compute types) VMs should be chosen (see https://aws.amazon.com/ec2/pricing/on-demand/). For a problem that use 20 Gbytes of memory, the *c5.4xlarge* is the smallest VM that will do the job. Max Number of threads should be set to zero allowing the maximum number of threads to be used which in this case is probably 16.

Note, TOPAS is threaded to a large extent, however, an excessive number of threads could slow down operation. For example, the large VM type of *c5.18xlarge* operating on the TEST_EXAMPLE\SINGLE-CRYSTAL\PN_02_2-ADPS.INP (3970 parameters) produces the following as a function of number of threads:

# of Threads	*approximate_A* - 15 iterations		Full A matrix - one iteration	
	Time(s)	Gain	Time(s)	Gain
2	42.19	0.32		
4	22.28	0.60	186.98	0.36
8	8.41	1.59	61.93	1.09
16	4.11	3.25	31.65	2.12
32	2.77	4.82	17.92	3.75
48	2.89	4.62	15.18	4.43
64	2.95	4.53	13.71	4.91
70	3.06	4.37	13.73	4.90

The columns marked Gain are the times taken on a high-end laptop with 8 threads divided by the time taken on *c5.18xlarge*. The speedup due to number-of-threads is substantial up to about 32 threads. It is worth noting that TOPAS V7 for the *approximate_A* case is 1.9 times faster than V6.

7.17 … Unable to connect to TC-VMs after local computer restart

The IP address of the local computer may change when the local computer is powered off and restarted, or, when the connection to the internet changes. VMs created prior to the restart would therefore have an invalid local-computer-IP-address; communication with the VMs would therefore not be possible. This scenario is noticed when the *Refresh* or *'Run TCs on selected VMs'* options of the *'Virtual Machines'* tab is not responsive. In such a case it is necessary to instruct the VMs that the IP address has changed. This can be performed from the *Instances* of the *EC2 Dashboard*; from this screen click on the security group shown in the *'Security Groups'* column. This brings up details of the security group. Click on *Inbound* and then *Edit* and then change the Source to *My IP*, or,
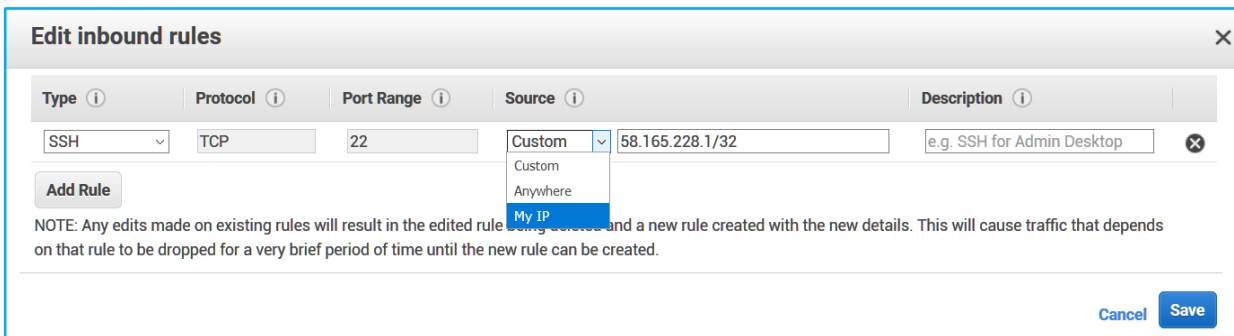
8. ..PROTEIN REFINEMENT

8.1..... Reading Protein Data Bank (PDB) CIF files

	Examples
[*pdb_cif_to_str_file* $file] ...	
[*pdb_ignode_adps* !E0]	CF-PROTEIN\2PVB-P212121\GEN.INP
[*pdb_cif_sites* $sites]	CF-PROTEIN\2PVB-P212121\MATCH.INP
[*pdb_cif_to_str* #0]	CF-PROTEIN\6Y84-C121\REFINEMENT.INP

Protein Data Bank (PDB) PDBx/mmCIF fles from https://www.rcsb.org/ can be downloaded and converted to INP format using *pdb_cif_to_str_file*. The operation is performed when *pdb_cif_to_str* is 1; on termination of refinement *pdb_cif_to_str* is set to 0 in the OUT file. The INP text generated is placed in the INP file after the *pdb_cif_to_str* keyword, or:

```
pdb_cif_to_str_file cif.cif
    pdb_ignode_adps  1
    pdb_cif_to_str   0
xdd_scr sf.cif
    lam lo 0.9096
    str
        scale @ 1
        a 51.03
        b 49.81
        c 34.57
        space_group P212121
        site ACE_C_0_1_HETATM    x  0.07354 y  0.35529 z  0.47637 occ C  1.00 beq  6.24
        site ACE_O_0_2_HETATM    x  0.06210 y  0.34246 z  0.50194 occ O  1.00 beq  7.96
        site ACE_CH3_0_3_HETATM  x  0.06198 y  0.35666 z  0.43651 occ C  1.00 beq  8.20
        site SER_N_1_4_ATOM      x  0.09557 y  0.36858 z  0.48319 occ N  1.00 beq  6.66
        site SER_CA_1_5_ATOM     x  0.10676 y  0.36880 z  0.52155 occ C  0.46 beq  8.09
        ...
        rigid
            point_for_site SER_N_1_4_ATOM    ux  -1.40900 uy  0.28011 uz  -1.21189
            point_for_site SER_CA_1_5_ATOM   ux  -0.83800 uy  0.29111 uz   0.11411
            point_for_site SER_CA_1_6_ATOM   ux  -0.70700 uy  0.20011 uz   0.04411
            ...
            Rotate_about_axies(@ 0 RX_, @ 0 RY_, @ 0 RZ_)
            translate tx @    6.28600 ty @   18.07889 tz @   17.91589
```

A rigid body is generated for each residue with coordinates set relative to its geometric center. Refinement can proceed on the generated INP text by setting the file name of *xdd_scr* to the name of the structure factor file 2PVB-SF.CIF, also downloaded from https://www.rcsb.org/. Running 2PVB\GEN.INP produces GEN.OUT; setting GEN.INP to GEN.OUT and running produces a fit.

pdb_cif_sites processes sites with names matching the site identifying string $sites. This can be used, for example, to extract all residues of the same type. The *translate* keywords of the rigid bodies can then be set to zero and the individual sites of the residues penalized such that sites of the same name are brought together; example INP text to do this is as follows:

```
        macro Match(s)
          {
              atomic_interaction s = R^2;
```

```
            ai_sites_1 s*
            ai_sites_2 s*
            ai_closest_N 1
            ai_only_eq_0
        penalty = s;
      }
   Match(LYS_N_)
   Match(LYS_CA_)
   Match(LYS_C_)
   Match(LYS_O_)
   Match(LYS_CB_)
   Match(LYS_CG_)
   …
```

Running example 2PVB\MATCH.INP produces the following showing overlay of LYS residues:
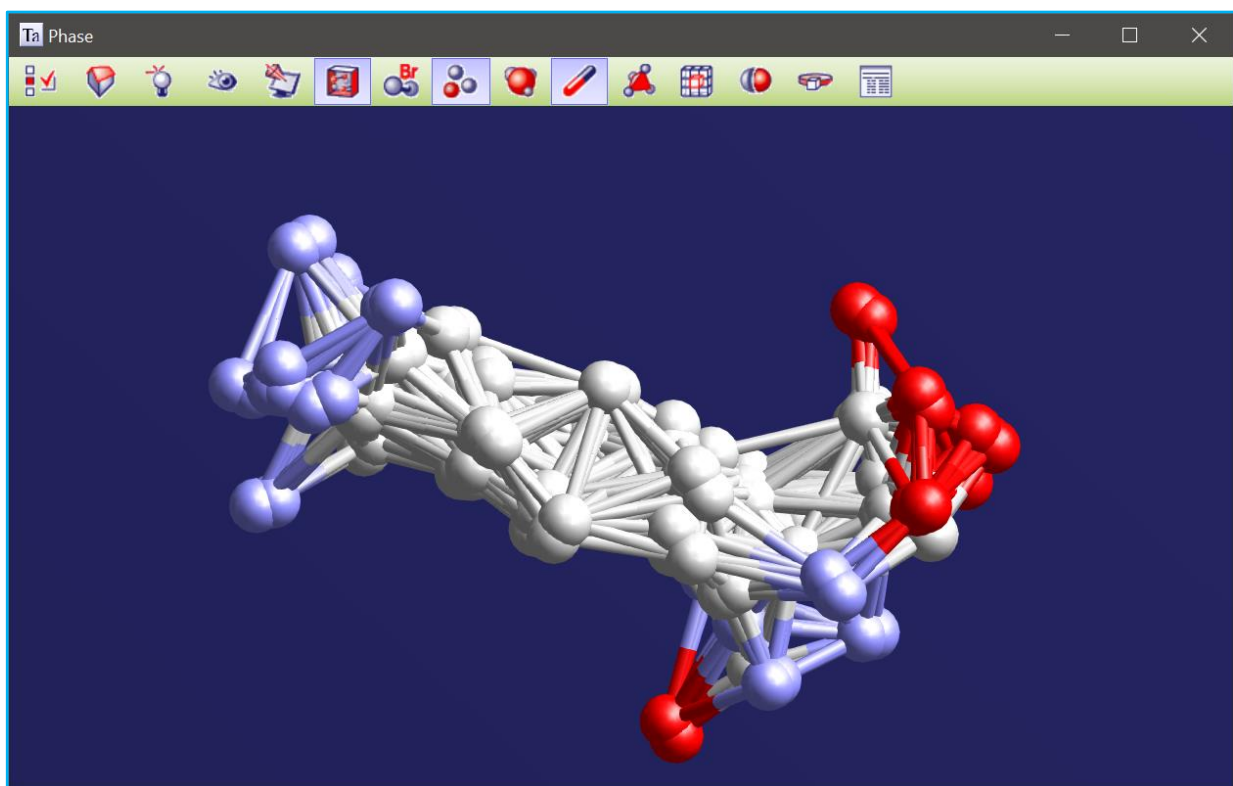
8.2 ..... Protein Refinement, 6y84, SARS-CoV-2 main protease

The structure factors and PDBx/mmCIF files for 6y84 can be downloaded from the PDB. To generate an initial INP file then create an INP file with the following (see 6Y84-C121\REFINEMENT.INP):

```
pdb_cif_to_str_file cif.cif
    pdb_ignode_adps  1
    pdb_cif_to_str 0
```

After refinement, the INP file can be updated with the structure generated from the CIF file. Refining on the updated INP file gives:
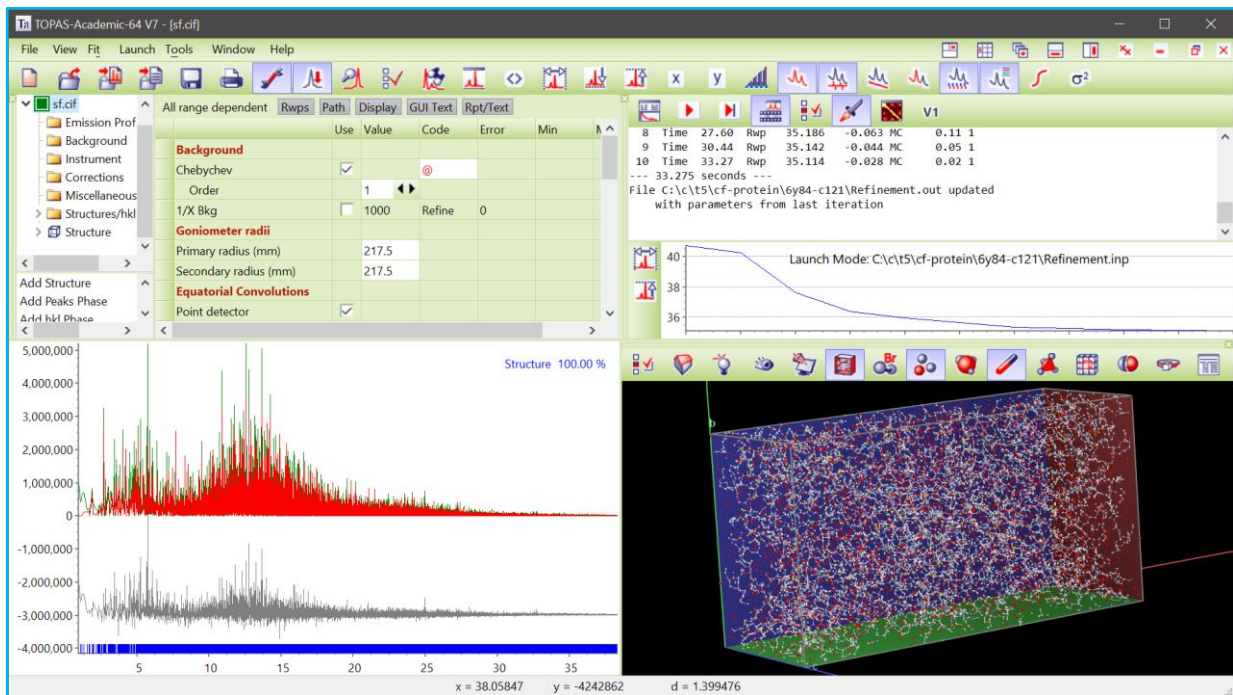
The refinement comprised 50348 unique reflections and 1826 parameters and the time to convergences was 33s on a laptop with all graphics operational. Restrains/constrains can of course be added.

9. ..SOLVING PROTEINS AT ATOMIC RESOLUTION

Include_Charge_Flipping

charge_flipping
    [*cf_plot_histo* !E]
    [*cf_plot_fit* !E]
    [*add_to_phases_of_non_weak_reflections* !E]...
    [*scale_flipped* !E]
    [*cf_percent_ED_ge_H* #]
    [*pick_atoms* $atom]...
        [*choose_from* !E]
        [*choose_to* !E]
        [*choose_randomly* !E]
        [*with_symmetry* !E]
        [*omit* !E]
        [*insert* !E]
        [*pick_fwhm* !E1]
        [*omit_fwhm* !E1]
        [*insert_fwhm* !E1]
    [*insert_atoms* {
        [*activate* !E1]
        [*in_cartesian*]
        [*insert_atom*]...
            [*x* !E][*y* !E][*z* !E][*occ* !E]
    }]...
    [*cf_set_phases* !E {
        #h #k #l #Re #im
    }]
    [*prm* N # *val_on_continue* !E]...

Macros in <u>CHARGE_FLIPPING.INC</u>

Examples
CF-PROTEIN\
    1A7Y-P1\SOLVE.INP
    2ERL-C2\SOLVE.INP
    1BYZ-P1\SOLVE.INP
    2KNT-P21\SOLVE.INP
    1AHO-P212121\SOLVE.INP
    4LZT-P1\SOLVE.INP
    1MC2-C121\SOLVE.INP
    1DY5-P21\SOLVE.INP
    2WFI-P212121\SOLVE.INP

    1HHZ-P3221\SOLVE.INP
    1C75-P212121\SOLVE.INP
    1B0Y-P212121\SOLVE.INP
    1CTJ-R3R\SOLVE.INP
    2PVB-P212121\SOLVEINP
    1CKU-P212121\SOLVE.INP
    1SWZ-P3221\SOLVE.INP
    5DA6-R32\SOLVE.INP

    1CTJ-R3R\1-ATOM.INP
    2PVB-P212121\1-ATOM.INP
    1C75-P212121\1-ATOM.INP
    5DA6-R32\1-ATOM.INP
    1CKU-P212121\1-ATOM.INP
    2WFI-P212121\1-ATOM.INP

    4LZT-P1\2-ATOMS.INP

The largest proteins ever solved *ab initio* at atomic resolution can be solved using modified charging flipping strategies. Difficult or large structures can be solved in minutes, rather than days using Amazon AWS Cloud computing. New/modified *charge_flipping* keywords are shown above. A single strategy does not solve all structures; A strategy successful on one structure is not necessarily successful on another. However, it will be shown that only two strategies can solve a large range of the most difficult structures. New keywords allow for a variety of strategies. *scale_flipped* scales flipped electron density (ED) charge; it is applied each charge-flipping iteration. *insert_atom* inserts atoms in the ED when *activate* is non-zero. *val_on_continue* for *prm*(s) are evaluated at the end of each charge-flipping iteration. *cf_percent_ED_ge_H* returns the percentage of ED pixels greater than 1 where the maximum of the ED is set to number of electrons in the heaviest atom defined by *f_atom_type*. Values less than 1 often signal a Uranium atom situation where a single ED peak dominates. *cf_percent_ED_ge_H* is displayed during charge flipping in the Fit Dialog.

When *cf_set_phases* is non-zero, the phases for the family of reflections (#h, #k, #l) are set to the phase corresponding to #Re and #Im. *cf_set_phases* is useful when phases are known or for

setting origin defining phases; for triclinic structures, three origin defining phases are possible. Additionally, intensities of the reflections are scaled by the value evaluated by *cf_set_phases*.

Table 9-1 show difficult benchmark structures, as listed by Elser *et al.* (2017) and Burla *et al.* (2011), that have been solved *ab initio*; see corresponding SOLVE.INP files for details. It is best to do preliminary investigations on the local computer (non-Cloud) to determine which strategy might work best. Once a strategy is chosen, INP files can be fed to the Cloud for rapid structure solution. Up to 500 spot instance Virtual Machines (VMs) are easily obtained on the Amazon AWS system in Australia at a cost of ~0.035 USD cents per VM per hour, or, 3.40 USD per hour for 100 machines. These prices are Amazon AWS dependent. Prices are shown prior to the creation of the VMs. The times shown in Table 9-1 can be easily doubled when one considers the preliminary analysis taken to arrive at the appropriate strategy. Typically, strategies are tried on the local computer before migrating the problem to the Cloud. Also, the structure solution process is normally halted after the first solution is found; for the investigative purposes, however, the structures in Table 9-1 were each solved at least 5 times. The two strategies mentioned in Table 9-1 are:

```
' S0 strategy
fraction_reflections_weak 0.5   add_to_phases_of_weak_reflections 90
fraction_density_to_flip  0.9   scale_flipped 0.6
```

S0 seems to work well for large structures with a relatively heavy atom. Non-triclinic structures with symmetry seems to succumb to the S1 strategy, or:

```
' S1 strategy
fraction_reflections_weak 0.5   add_to_phases_of_weak_reflections = Rand(-180, 180);
fraction_density_to_flip  0.97 scale_flipped 0.2
pick_atoms *
   pick_fwhm 3
   choose_randomly = If(Mod(Cycle_Iter, 50), 0, 10);
   with_symmetry 1
   insert 10        ' Increase if the most dominant atom does not change
symmetry_obey_0_to_1 0.25 find_origin 0
flip_regime_2 = Sine_Wave(10/4,-2,2,10); ' Used when there's not enough perturbation
```

S1T extends the S1 strategy with the addition of the tangent formula, or, the inclusion of:

```
Tangent(0.5, 30)
```

Table 9-1. *Ab initio* structure solution strategies. Time indicates time to solution on average. Each structure was solved at least 5 times. *Num_VMs* greater than 8 refers to the number of VMs used on the Cloud; *Num_VMs*=9 corresponds to an 8 core local computer (a laptop). Cost corresponds to the average Cloud cost to a solution using the strategy indicated.

Solved	PDB code	Space group	N/Z	d_{min} (Å)	Time (min)	Num VMs	Cost USD	Strategy	N_p
yes	1a7y	*P*1	270	0.94	0.1	8	–	S0	–
yes	2erl	*C*2	303	1.00	1	200	0.10	S1	8
yes	1byz	*P*1	408	0.90	1	200	0.10	S0	–

yes	2knt	$P2_1$	460	1.20	16	200	2.00	S1T	7
yes	1aho	$P2_12_12_1$	500	0.96	1	200	0.10	S1	8
No	1w7q	$P6_5$	828	1.10	>240	200	>28	S0,S1	4
yes	4lzt	$P1$	1183	0.95	2	8	-	S1	10
yes	1mc2	$C2$	1254	0.80	2	200	0.20	S1	10
yes	1dy5	$P2_1$	1894	0.87	1	500	1.40	S1	30
yes	2wfi	$P2_12_12_1$	1920	0.75	18	500	5.10	S1	15
yes	1hhz	$P3_22_1$	354	0.99	7	200	1.00	S1	6
yes	1c75	$P2_12_12_1$	1184	0.92	1	8	-	S0	-
yes	1b0y	$P2_12_12_1$	837	0.93	1	8	-	S0	-
yes	1ctj	$R3:R$	918	1.10	1	200	0.10	S1	4
yes	2pvb	$P2_12_12_1$	1096	0.91	3	200	0.35	S1	5
yes	1cku	$P2_12_12_1$	1599	1.20	1	200	0.15	S1	8
yes	1swz	$P3_22_1$	1254	1.06	50	200	5.80	S1	15
yes	5da6	$R32$	1390	1.05	5	500	1.40	S1	15

PDB code	Reference
1a7y, 2erl, 1byz, 2knt, 1aho, 1w7q, 4lzt, 1mc2, 1dy5, 2wfi	Elser & Lan (2017)
1hhz, 1c75, 1b0y, 1ctj, 2pvb, 1cku, 1swz	Burla et al. (2011)
5da6	Mooers (2016)

PDB codes 1b0y, 1ctj, 1c75 and 1cku are easily solved (a few minutes) on a laptop using the S0 strategy. 2knt uses the tangent formula due to its relatively low-resolution data (1.2Å) as well as its relatively small number of non-hydrogen atoms in the asymmetric unit. 1w7q is a light element structure that was not solve *ab initio* after more than four hours. *flip_regime_2* of S1 introduces perturbation and it should be used for cases where there the ED seems quiet during the charge flipping process; decreasing the absolute value of *flip_regime_2* reduces perturbation. In the case of 1cm2, *flip_regime_2* was set to oscillate between –1 and 1. Larger values clearly shows too much perturbation in the ED.

Graphically inspecting the ED or looking at the (%ED > H) output on the local can be used to determine if there's too little or too much perturbation, during charge flipping. (%ED > H) should typically range from 1 to 5. For example, setting *fraction_reflections_weak* to 0.9 results in too much perturbation. Or, using the Tangent formula macro on *P*1 structures, without the mitigation strategy of *Fix_Uranium_3*, results in too little perturbation resulting in uranium atom solutions. The value set for *Fix_Uranium_3* should be just high enough to prevent Uranium atom solutions; a value of 1 seem to work in most cases. The number used for *insert* of *pick_atoms* should be just high enough to change the position of the highest intensity ED peak every 40 to 50 iterations as defined by *choose_randomly*; note *pick_atoms* is executed when *choose_randomly* is greater than zero. *add_to_phases_of_weak_reflections*=90 results in a shifting origin and it should not be used with *symmetry_obey_0_to_1* ; the latter prevents origin shifting.

add_to_phases_of_weak_reflections should be set to Rand(-180,180) instead of 90 when using *symmetry_obey_0_to_1*. Further structure solution tips are:

- Try the simple S0 strategy first for number of atoms less than about 300.
- If a heavy atom is present, then try S0.
- Inspect the ED graphically; if it does not show distinct atoms after a few iterations then change strategy.
- Use S1 for large difficult structures.
- Try the tangent formula when the number of non-hydrogen atoms in the asymmetric is less than ~500 atoms. The tangent formula reduces perturbation allowing lower resolution structure to be solved.

The range of convergence of structure factor phases can be investigated by loading optimum structure factor phases values, using *set_initial_phases_to*, and then adding to the optimal phases using *randomize_initial_phases_by*. High resolution data can have their optimal phases changed by an amount of 0.96*Rand(-180,180) whilst still being able to solve the structure within a few dozen charge flipping iterations. Most of the SOLVE.INP examples contain the following for investigating this range of convergence:

```
#if (0)
    set_initial_phases_to optimal.fc
    randomize_initial_phases_by = Rand(-180, 180) 0.9;
#endif
```

9.1..... Ab initio solution of triclinic 4lzt

PDB code 4lzt comprises 1183 non-hydrogen atoms in the unit cell and is considered difficult to solve, see Elser *et al*., 2017. 4lzt contains 10 Sulphur atoms and these are considered moderately heavy. If we were to insert ED peaks at positions corresponding to the highest two peaks of the optimum electron density, then charge flipping finds a solution and within a few iterations; 4LST\2-ATOMS.INP demonstrates this where an ED starting with the two highest optimal peaks, inserted using *insert_atoms*, produces and R-factor plot of:
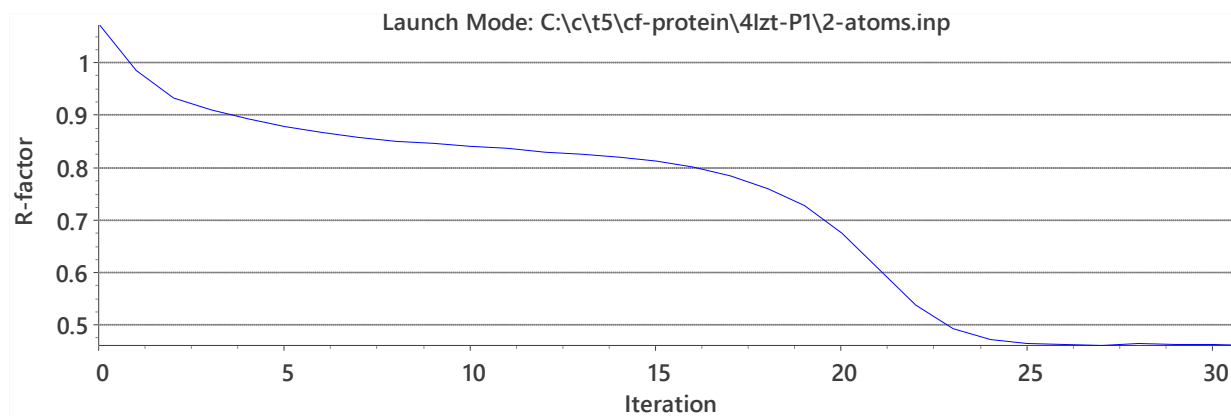
In fact, any two of the five highest peaks produce similar R-factor plots. However, these optimal ED peak positions are unknown. The strategy that works therefore involves picking an atom

randomly out of the 10 largest peaks in the electron density and setting the picked atom to a large density. The INP file looks like:

```
fraction_reflections_weak 0.5
    add_to_phases_of_weak_reflections = Rand(-180, 180);
fraction_density_to_flip 0.97
    scale_flipped 0.2
pick_atoms *
    pick_fwhm 5 omit_fwhm 1 insert_fwhm 1
    choose_randomly = If(Mod(Cycle_Iter, 50), 0, 10);
    insert 10
Fix_Uranium_3(0.5)
ATP(1000, 1) ' Totally randomize phases after 1000 iterations
```

pick_atoms picks atoms with a FWHM of 5 Å, as defined by *pick_fwhm*; this relatively large value ensures that the picked atoms are approximately 5 Å apart. Once picked, *pick_atoms* removes the atoms with a FWHM as defined by *omit_fwhm*, and then inserts atoms with a FWHM of *insert_fwhm*. A solution of 4lzt takes a minute or two on a laptop computer and a typical R-factor plot looks like:
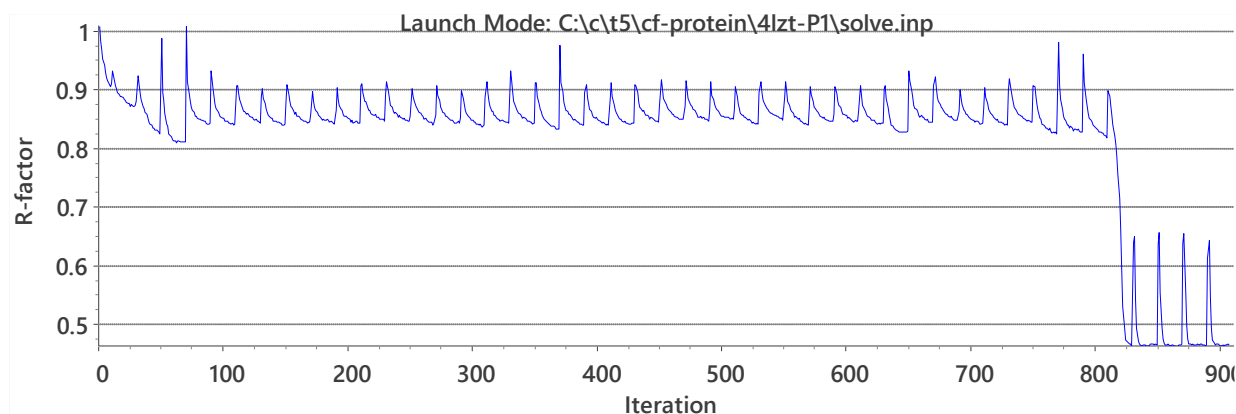
9.2..... Solution of non-triclinic lattices using a known atomic position

Large non-triclinic structures with many origins are difficult to solve. However, because of symmetry, non-triclinic structures can often be solved when the position of a single atom is known within the ED. Atoms can be inserted in the ED using *insert_atoms*; for PDB code 2wfi we have:

```
charge_flipping
    cf_hkl_file sf.cif ' Structure fact file from PDB
    space_group P212121
    a  37.544  b  65.144  c  69.680
    fraction_reflections_weak 0.5
        add_to_phases_of_weak_reflections = Rand(-180, 180);
    fraction_density_to_flip 0.97
        scale_flipped 0.2
    symmetry_obey_0_to_1 0.25 find_origin 0
    macro Occ_0 { 100 }
    insert_atoms {
        activate = Mod(Cycle_Iter, 100) == 0;
        load insert_atom x y z occ  {
            0.72697 0.77709 0.11312  100 ' Position of known atom
        }
```

```
        }
```

The *x*, *y*, *z* coordinates of *insert_atom* can be in Cartesian coordinates using the *in_cartesian* key-word at the *insert_atoms* level. The use of *symmetry_obey_0_to_1* often assists in solution deter-mination for non-triclinic structures. 2WFI can be solved *ab initio*, however it can be easily solved if the position of one atom was known as seen by tunning 2WFI-P212121\1-ATOM.INP; it gives and R-factor plot that looks like:
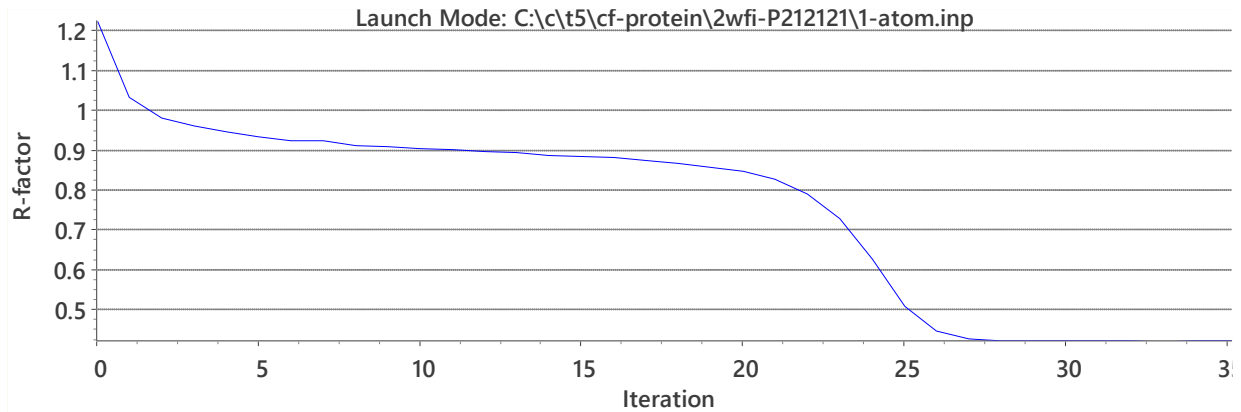
The OpenGL plot shows the solution as follows:
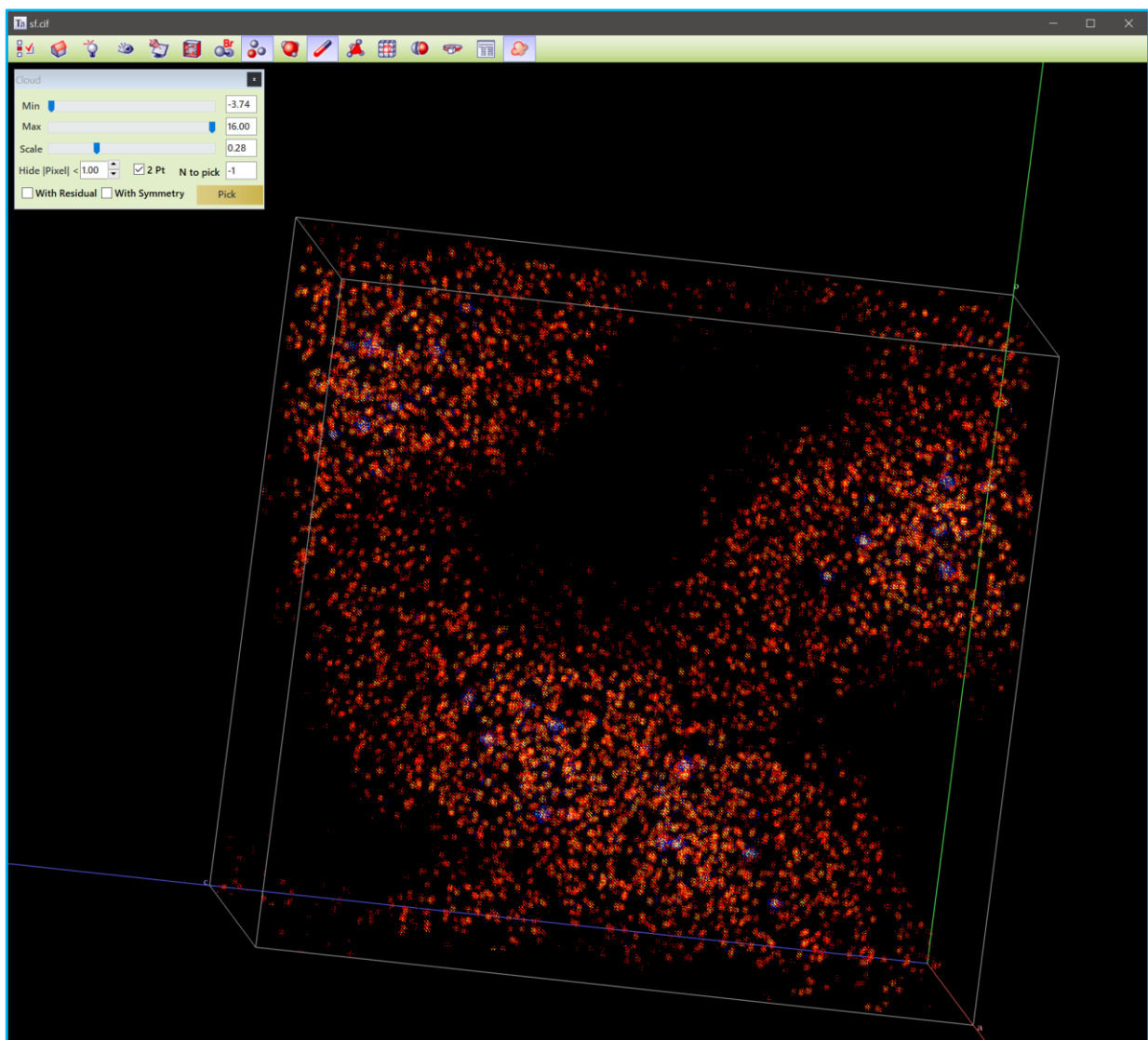
Using any one of the first six highest optimal ED peaks results in a solution. Many structures can be solved from knowing the position of just one atom. 1-ATOM.INP files, similar-to the 2wfi case, are given for 1ctj, 2pvb, 1c75, 5da6, 1cku, 2wfi.

9.3..... Ab initio solution of 5da6 in space group *R*3$_2$

PDB code 5da6 comprises 1390 atoms in the asymmetric unit. Placing an ED peak at any of its potassium sites result in the correct solution (see 5DA6-R32\1-ATOM.INP). 5da6 can also be solved *ab initio* using the following INP file (see 5DA6-R32\SOLVE.INP):

```
charge_flipping
    cf_hkl_file sf.cif ' Structure factor file from PDB
    space_group R32
    a  42.890  b  42.890  c  266.936  ga 120.00
    fraction_reflections_weak 0.5
        add_to_phases_of_weak_reflections = Rand(-180, 180);
    fraction_density_to_flip 0.97
        scale_flipped 0.2
    symmetry_obey_0_to_1 0.25 find_origin 0
    pick_atoms *
        pick_fwhm 5 omit_fwhm 1 insert_fwhm 1
        choose_randomly = If(Mod(Cycle_Iter, 50), 0, 15);
        insert 10
    flip_regime_2 = Sine_Wave(50 / 4, -2, 2, 50);
    ATP(1000, 1) ' Randomize all phases every 1000 iterations
```

It takes approximately six hours on average to solve 5da6 using the above INP file on an 8-core laptop computer. This time is reduced to 5 minutes on the Cloud where the INP file is run simultaneously on 500 VMs. The best solution on each VM computer or the best solution overall can be viewed during the process. A typical Cloud run looks like:
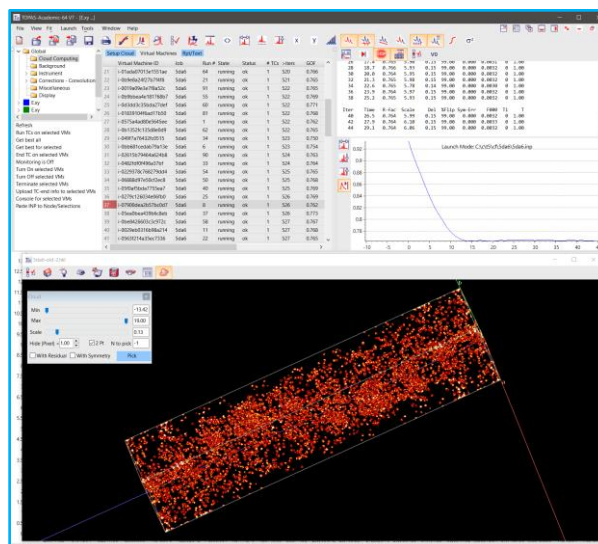
10. DECONVOLUTION

[*A0_matrix_is_constant*]
[*create_pks_name* $a_name]
[*create_pks_fn* $fn_name]
	<u>*Examples*</u>
	TEST_EXAMPLES\DECONVOLUTION\
	    PBSO4-DECON.INP
	    SIM-CALC.INP
	    SIM-DECON.INP

The deconvolution method of [a]Coelho (2018) has been implemented; it uses three macros found in TOPAS.INC, *Deconvolution_Init*, *Deconvolution_Bkg_Penalty* and *Deconvolution_Intensity_Penalty*. The method refines on linear parameters only; these linear parameters are peak intensity and background parameters; their derivatives are unchanging and hence the A_0 matrix is unchanging. The keyword *A0_matrix_is_constant* informs the program that only linear parameters are being refined and hence the A_0 matrix is calculated only once. Attempts to use *A0_matrix_is_constant* with *quick_refine*, *approximate_A*, *chi2* or with refinement of non-linear parameters results in an error.

create_pks_name is a *xo_ls* dependent keyword that creates a peak at each step along the x-axis with peak intensity parameter names starting with the string $a_name. Peaks are not created if peaks already exist for the *xo_ls* phase. If the '$' character is placed immediately after *create_pks_name* and if *create_pks_name* is within a macro then the output from *create_pks_name* is placed after the macro. *create_pks_fn* additionally appends a *penalty* to each peak with the *penalty* being written in terms of a function called fn_name. The OUT file is updated with peaks which looks something like:

```
xo 5.00 I a25_ 0.00217` penalty = dfn(5,a25_,a26_);
xo 5.02 I a26_ 0.00000` penalty = dfn(5.02,a26_,a27_);
xo 5.04 I a27_ 0.00000` penalty = dfn(5.04,a27_,a28_);
```

The dfn function takes arguments of x-axis position of the peak and two intensity parameter names, one at the x-axis position and the other at the next x-axis position. These keywords and functions are used in macros in the following manner:

```
Deconvolution_Init(0.5)
xdd …
   Deconvolution_Bkg_Penalty(0.5)
   xo_Is
      Deconvolution_Intensity_Penalty(a, afn)
```

The deconvolution process comprises three separate refinement runs. 1) Fitting the peaks to the diffraction pattern with peak shapes fixed to expected peak shapes, 2) creating a calculated pattern with a chosen peak shape, typically a peak shape comprising specimen contributions, and 3) a final run to produce a deconvoluted pattern with noise. The PBSO4-DECON.INP example is ready to run, it can be used as a template for other deconvolution processes, it is defined as:

```
#define DO_REFINEMENT_      ' Step 1
'#define DO_SPECIMEN_OUT_   ' Step 2
'#define DO_FINAL_DECON_    ' Step 3
macro Data_File { Pbso4 }
```

```
    #ifdef DO_FINAL_DECON_
      RAW(..\##Data_File) ' load for comparison purposes
     xdd Data_File##-decon-specimen.xy
         x_calculation_step 0.025
         user_y d1 Data_File##-decon-specimen.xy
         user_y d2 Data_File##-diff.xy
         fit_obj = d1 + d2;
         Out_X_Ycalc(Data_File##-decon-final.xy) ' Final deconvoluted pattern
    #else
        Deconvolution_Init(0.5)
        RAW(..\##Data_File)
         start_X 15
         bkg @ 0 0 0 0 0 0 0
         Deconvolution_Bkg_Penalty(0.5)
         'LP_Factor(17) ' Do not include when doing deconvolution
         CS_L(262.73494)
         Strain_L(0.03785)
         #ifdef DO_SPECIMEN_OUT_
             iters 0
             CuKa1(0.0001)
             Out_X_Ycalc(Data_File##-decon-specimen.xy)
         #else ' DO_REFINEMENT_
             Out_X_Difference(Data_File##-diff.xy)
             CuKa5(0.0001)
             Radius(173)
             Full_Axial_Model(10, 10, 10, 4.13679, 4.13679)
             Divergence(1)
             Slit_Width(0.2)
         #endif
         xo_Is
             Deconvolution_Intensity_Penalty(a, dfn)
    #endif
```

Background should be less than all observed data and it should be graphically inspected during step 1. Background can be reduced by decreasing the *c* parameter of the *Deconvolution_Bkg_Penalty* macro; this parameter can range from 0.05 to 1. If the bases of the peaks are not fitting well, then the background is still too high. Step 1 and 2 produces output XY files which are then used in step 3. The exclusion of *LP_Factor*, and similar peak scaling parameters, is important as peak intensities are used in a penalty inside the *Deconvolution_Intensity_Penalty* macro. The deconvolution process can be used for all types of data including neutron TOF; step (1) takes approximately 10 to 30 seconds on present laptops; steps (2) and (3) takes a trivial amount of time (< 1s). The deconvolution macros are as follows:

```
    macro Deconvolution_Init(c) {
       process_times
       A0_matrix_is_constant      ' All parameters are linear
       penalties_weighting_K1 = c; ' A value of 0.5 seems sufficient
       save_best_chi2             ' We want best Chi2; not best Rwp
       chi2_convergence_criteria 1e-5
       continue_after_convergence ' ~100 iterations is typically sufficient (~20s)
       pen_weight 1               ' Override the default
    }
    macro Deconvolution_Intensity_Penalty(i_name, fn_name) {
       fn fn_name(x, a0, a1) = (a0 - a1)^2 / ((a0 + a1) Yobs_at(x) + 1e-6);
       default_I_attributes 1e-6 min 0 val_on_continue = Val Rand(0.99, 1.01);
```

```
    create_pks_fn fn_name
    create_pks_name $ i_name
  }
  macro Deconvolution_Bkg_Penalty(& c, & w_min) {
    xdd_sum #m_unique pen = (Yobs - Get(bkg))^2 / Max(Get(bkg) Yobs, w_min^2);
    penalty = pen c;
  }
  macro Deconvolution_Bkg_Penalty(& c) { Deconvolution_Bkg_Penalty(c, 1) }
```

pen_weight over-rides the default; the default works but with slower convergence. Note, both the peak intensity and Bkg penalties are *Yobs* scale invariant where scaling of *Yobs* does not change the magnitude of the penalties relative to χ_0^2. Yobs_at is a new function that returns the value of *Yobs* at x. w_min in the *Deconvolution_Bkg_Penalty* macro allows for the setting of the expected minimum of *Yobs**Bkg; a value of 1 for counting statistics. For XYE files, where *Yobs* is small and where *SigmaYobs* used (tof data for example), then w_min should be reduced.

10.1 ... Deconvolution – Simulated pattern

A simulated pattern was created with noise using SIM-CREATE.INP and the instrument contribution deconvoluted using SIM-DECON.INP; the latter INP file looks like:

```
/* Three runs to produce the deconvoluted pattern.
   The name of the final deconvoluted pattern is:

       pbso4-decon-final.xy

   Define one at a time in the following:

       #define DO_REFINEMENT_    ' Run 1
       #define DO_SPECIMEN_OUT_ ' Run 2
       #define DO_FINAL_DECON_   ' Run 3
*/
#define DO_REFINEMENT_       ' Step 1
'#define DO_SPECIMEN_OUT_    ' Step 2
'#define DO_FINAL_DECON_ ' Step 3, Clear the GUI first

macro Data_File { Sim }
#ifdef DO_FINAL_DECON_
   xdd Data_File##-calc-rand.xy ' load for comparision purposes
   xdd Data_File##-calc-narrow.xy
      user_y d1 Data_File##-decon-specimen.xy
      user_y d2 Data_File##-diff.xy
      fit_obj = d1 + d2;
      Out_X_Ycalc(Data_File##-decon-final.xy)
#else
   Deconvolution_Init(0.5)
   xdd Data_File##-calc-rand.xy
      bkg @  259.381081  89.8339877  31.6429117 -34.4743462
             34.3097757 -55.7270435  30.631573
      Deconvolution_Bkg_Penalty(0.1)

      /* Specimen */
      CS_L(300)
      CS_G(300)
      Strain_L(0.05)
```

```
        Strain_G(0.05)

        #ifdef DO_SPECIMEN_OUT_
            iters 0
            CuKa1(0.001)
            Out_X_Ycalc(Data_File##-decon-specimen.xy)
        #else ' DO_REFINEMENT_
            num_cycles 20
            Out_X_Difference(Data_File##-diff.xy)

            /* Instrument */
            CuKa2(0.001)
            Radius(217)
            Full_Axial_Model(12, 12, 12, 2.3, 7)
            Divergence(1)
            Slit_Width(0.1)
            Absorption(60)
        #endif
        xo_Is
            Deconvolution_Intensity_Penalty(a, dfn)
    #endif
```

The following figure is the deconvoluted pattern (green line, bottom plot) compared with the expected deconvoluted pattern (red line on top of green line). The top plot (blue line) is the original simulated pattern with noise and without noise (red line on top of blue line).
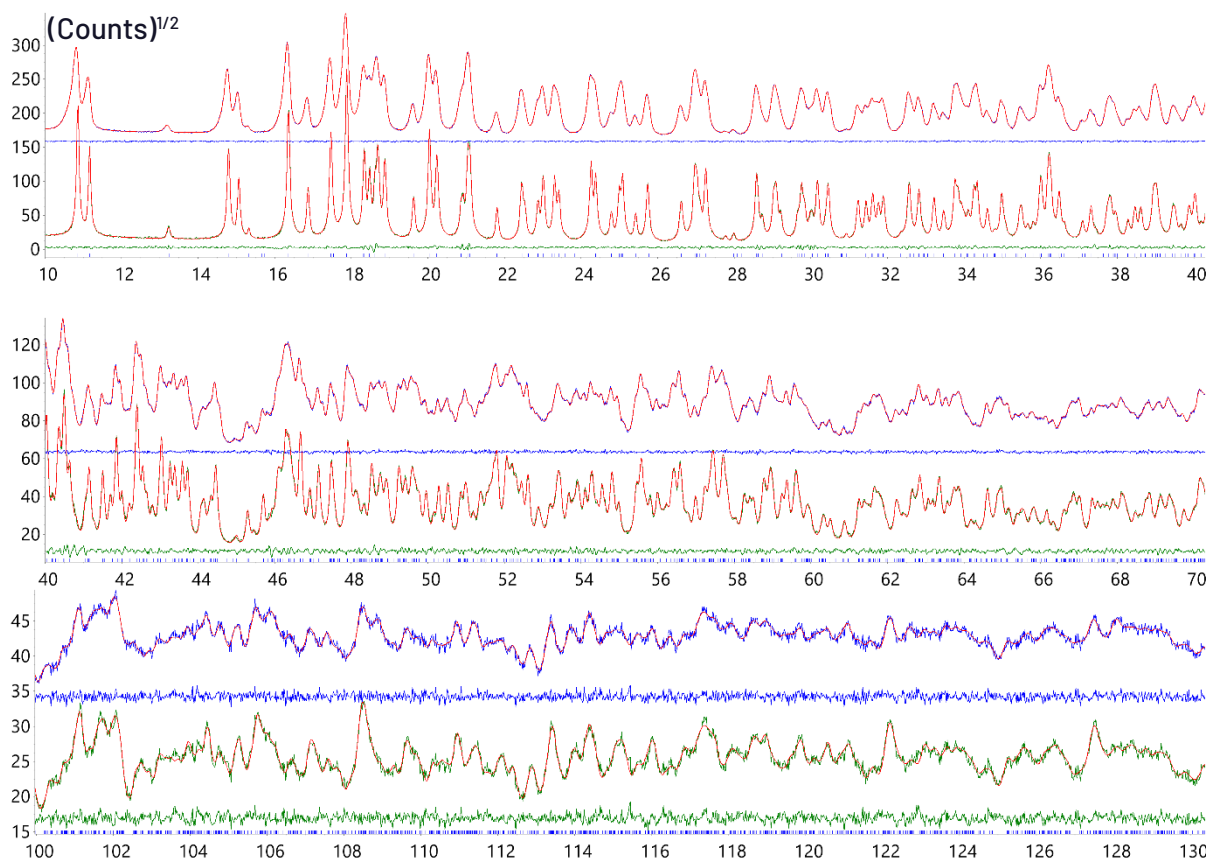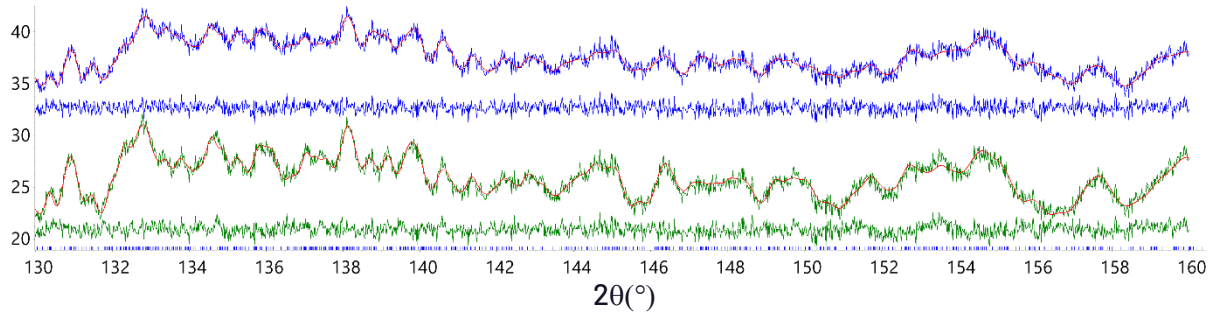
Parameter errors determined from refinement using the deconvoluted pattern are almost identical to errors produced using the original pattern, see [a]Coelho (2018).

11. PDF-GENERATION

	Examples

[*xdd*...]
    [*rebin_with_dx_of* !E]
    [*pdf_generate* {
        [*dr* !E]
        [*r_max* !E]
        [*gr_sst_file* = "File";]
        [*hat* !E [*num_hats* !E]
        [*gr_to_fq* !E]
    }]
	Examples
	TEST_EXAMPLES\PDF\GENERATE\
	    FULLERENE\DECON.INP
	    LIFEPO4\DECON.INP
	    SILICON\DECON.INP
	    TUNGSTEN\DECON.INP

PDF generation comprises an inverse Sine transform operating on an ideal diffraction pattern where background is absent, atomic scattering factors constant, and 2θ and peak shapes are symmetric. The task therefore becomes one of correcting real data such that it matches an ideal pattern as closely as possible. The corrections include determining a background, atomic scattering factors (if X-ray data), removing Lorentz polarization and removing asymmetry from peak shapes. To generate the PDF, a deconvolution process similar-to that described in section 10 is used. It allows for corrections in reciprocal space of peak asymmetry, instrument and emission profile aberrations, Lorentz polarization and atomic scattering factors corrections. The process comprises two operations described in a single INP file; these operations are:

- 0) Fit to the reciprocal space diffraction pattern - (Operation 0)
- 1) Generate $G(r)$ - (Operation 1)
  - 1.0) Generate ideal pattern Ideal (2θ) from the parameters determined in step 0.
  - 1.1) Convert Ideal (2θ) to Q space to form *Ideal*(Q).
  - 1.2) Fit a polynomial to *Ideal*(Q) and save $F(Q) = Ideal(Q) - Poly$)
  - 1.3) Generate $G(r)$ from $F(Q)$

Each operation requires running the INP file once. Steps 1.0 to 1.3 of operation 1 is performed with *num_runs* set to 4.

11.1 ... PDF-Generating - LiFePO4

Fitting to the pattern, operation 0, follows the deconvolution process of [a]Coelho (2018). Lattice parameters are not required. A peak is laid down at each data point of the pattern together with a background and appropriate penalty functions. Approximate peak shapes from a preliminary peak fitting analysis, using a 'standard' for example, is recommended; once determined peak shapes are not refined. The data entry part of a typical INP file (see LIFEPO4\ DECON.INP for example) is as follows:

```
Include_PDF_Generate
'-------------------------------------------------
'          START USER INPUT SECTION
'-------------------------------------------------
macro Data_File         { LFP_0-8Kcap_AgFGM_2x4soll_Eiger1D_8h.xy }
macro Capillary_Scan    { capillary.xy }
macro Capillary_Rebin   { 0.1 } ' Smooth the capillary scan. Zero means no smoothing.
```

```
#prm operation = 0; ' Set to 0 to fit to reciprocal space data
                    ' Set to 1 to generate F(Q) and G(r)
                    ' Set to 2 to fit structure to G(r)
#prm use_narrow_peak_shape  = 1; ' A 0 means use full peak shapes in generating G(r)
'--------------------------------------------------
' Inputs for reciprocal space fit, operation = 0
macro & Average_f          { f0_Li + f0_Fe + f0_P + 4 f0_O } ' formula of unit cell
#prm lab_no_monochromator  = 1; ' Set to 1 if using Laboratory instrument.
#prm use_Xo_Is_phase       = 1; ' Set to 0 if not fitting peaks
#prm use_bkg_penalty       = 1;
#prm use_simple_bkg_penalty = 1; ' Set to 1 if counting statistics is not right,
                                 ' or, maybe when there's Fluorescence.
macro & Bkg_Weighting              { 1 }
macro & Intensity_Penalty_Weighting { 1 }
macro & Scale_Peaks   { 1 } ' Useful if capillary absorption is inhibiting fitting.
macro & Scale_Yobs_By { 1 } ' Useful if data does not obey counting statistics.
prm pc0  1   ' Poly_Capillary coefficients; comment out if
prm pc1  0   ' not using Capillary as background.
inp_text fluorescence_bkg
   {
      bkg @ 3.49160163` -0.96682842`  0.292687899`
   }
inp_text fit_extra
   {
      penalty = 10000 (Bkg_at(X2) + (pc0 + pc1) Value_at_X(cap_, X2) - Yobs_at(X2))^2;
   }
macro Start_X       { 2.4  }
macro Finish_X      { 103  }
macro Step_X        { 0.02 } ' Set to zero to use measured step size.
                             ' Set to non-zero if scale_yobs_by is used.
                             ' Set to non-zero if unequal x-axis steps.

'--------------------------------------------------------------
' Inputs for generating F(Q), operation = 1
#prm poly_fq = 7;  ' Number of parameters for Poly when fitting Poly to F(Q).
                   ' View F(Q) plot, it needs to look right.
macro & Qmin                { 0.1 }
macro & Qmax                { 17.5 }
macro & Soper_Lorch_Constant  { 0 } ' best not to use
macro & Exp_Constant          { 0 } ' best not to use
macro & Lorch_Constant        { 0 } ' best not to use
inp_text fq_poly
   {
      bkg @  0 0 0 0 0 0 0  0
   }
macro FQ_Bkg_Penalty
   {
      weighting = If(X > (X2 - 1), 10, 1); ' Weigh the F(Q) data more at Qmax
      penalty   = Bkg_at(X1)^2;            ' Restrain F(Q=0) to 0
   }
'--------------------------------------------------------------
' Inputs for generating G(r) from F(Q), operation = 1
macro R_Max        { 100 }
macro dR           { 0.01 }
macro Num_Hats     { 3 } ' Best smoothing function for speed and accuracy
macro & Hat_Size   { 4.4934 / Qmax }
'--------------------------------------------------------------
' Reciprocal space peak details, operation = 0
macro Full_Emmision_Profile
```

```
    {
        lam ymin_on_ymax 0.001
            la 1 lo 0.5609 lg 1e-6
            la 0.55150 lo 0.5649441 lg 1e-6
    }
macro Deconvoluted_Emmision_Profile
    {
        lam ymin_on_ymax 0.0005 la 1 lo 0.5609 lg 1e-6
    }
macro Full_Peak_Shape_Specimen
    {
        CS_G(, 70)
        CS_L(, 45)
        Strain_L(, 0.042)
        Strain_G(, 0.42)
    }
macro Full_Peak_Shape
    {
        Full_Peak_Shape_Specimen
        Full_Axial_Model(10,10,10, 2.3, 5.73430)
    }
macro Deconvoluted_Peak_Shape
    {
        Deconvoluted_Emmision_Profile
        #if (use_Xo_Is_phase == 0)
            ' Using (Yobs - background); ie. no peak shape
        #elseif (use_narrow_peak_shape)
            ' Use Narrow peak shape
            ZE(, -0.00730929318) ' Set to negative of Rietveld fit
            gauss_fwhm 0.05
        #else ' Use Full peak shape specimen
            Full_Peak_Shape_Specimen
            ZE(, -0.00730929318) ' Set to negative of Rietveld fit
        #endif
    }
macro & LP_Factor_
    {
        #if (lab_no_monochromator)
            (1 + Cos(X Pi/ 180)^2)
        #endif
        1 / (Sin(X Pi/360)^2 Cos(X Pi/360)) ' Lorentz factor
    }
'--------------------------------------------------
'              END USER INPUT SECTION
'--------------------------------------------------
Include_PDF_Generate_Common
'--------------------------------------------------
#if (And(use_Xo_Is_phase, Run_Number == 0, Or(fit_to_data, generate_fq_gr_from_fit)))
    xo_Is
        PDF_Generation_Intensity_Penalty(a,dfn, Intensity_Penalty_Weighting, Scale_Peaks)
#endif
'--------------------------------------------------
```

The user needs to input data such as the name of the data files etc... It is best to create a new directory for each data file. The PDF-GENERATE.INC file, included using the *Include_PDF_Generate* macro, contains PDF generation specific macros. *Capillary_Scan* is the name of the file

corresponding to a scan of the empty capillary sample holder. Typically, the capillary scan is collected in a short time leading to poor counting statistics; Capillary_Rebin can therefore be used to smooth the capillary scan. Setting the #prm called *operation* to 1 instructs the program to perform the fitting process. Setting *use_narrow_peak_shape* to 1 result in narrow peaks being used in the generation of the *Ideal*(2θ) (operation 1.0); this removes peak broadening as a function of 2θ.

11.1.1 ....Operation 0 – Fitting peaks to the diffraction pattern

If *use_Xo_ls_phase*=0 then no peak fitting is performed and hence no deconvolution; the ideal pattern is created using (Yobs – Ycalc)/ (LP_Factor <f>), where Ycalc in this case is the background function. Also, *use_simple_bkg_penalty* should also be set to 1. When *use_Xo_ls_phase*=1, peaks are fitted. The program internally creates peaks and places them at the position of the *xo_ls* phase. *lab_no_monochromator*=1 instructs the program that the data is from a Laboratory instrument without a monochromator. Background is described as follows:

> *Background = Poly_Capillary * Capillary_Scan + Poly_Fluorescence*

Poly_Capillary is a 1st order polynomial with coefficients defined by the pc0 and pc1 parameters. *Poly_Fluorescence* is also a nth order Chebyshev polynomial with coefficients defined by the user at the *inp_text* fluorescence_bkg {} construct; set this construct to blank when not using. LiFePO4 fluoresces and its best to use the smallest number of *bkg* parameters whilst producing a good background fit. In the case of LiFePO4, the high angle peaks seem to vanish. This means that the background should be almost equal to Yobs at the highest angle X2. Such a condition can be enforced using a penalty as shown in the *inp_text* called *fit_extra*. The penalty describes the following:

> (*Poly_Capillary*(X2) * *Capillary_Scan*(X2) + *Poly_Fluorescence*(X2) – Yobs_at(X2))^2

X2 is the reserved parameter name corresponding to the end of the diffraction pattern. *Poly_Capillary* at X2 is simply (pc0 + pc1), see the X0_ macro in PDF-GENERATE_COMMON.INC, and *Poly_Fluorescence*(X2) corresponds to Bkg_at(X2). The *penalty* therefore looks like:

```
inp_text fit_extra
   {
      penalty = 10000 (Bkg_at(X2) + (pc0 + pc1) Value_at_X(cap_, X2) - Yobs_at(X2))^2;
   }
```

The fit for LiFePO4 looks like:
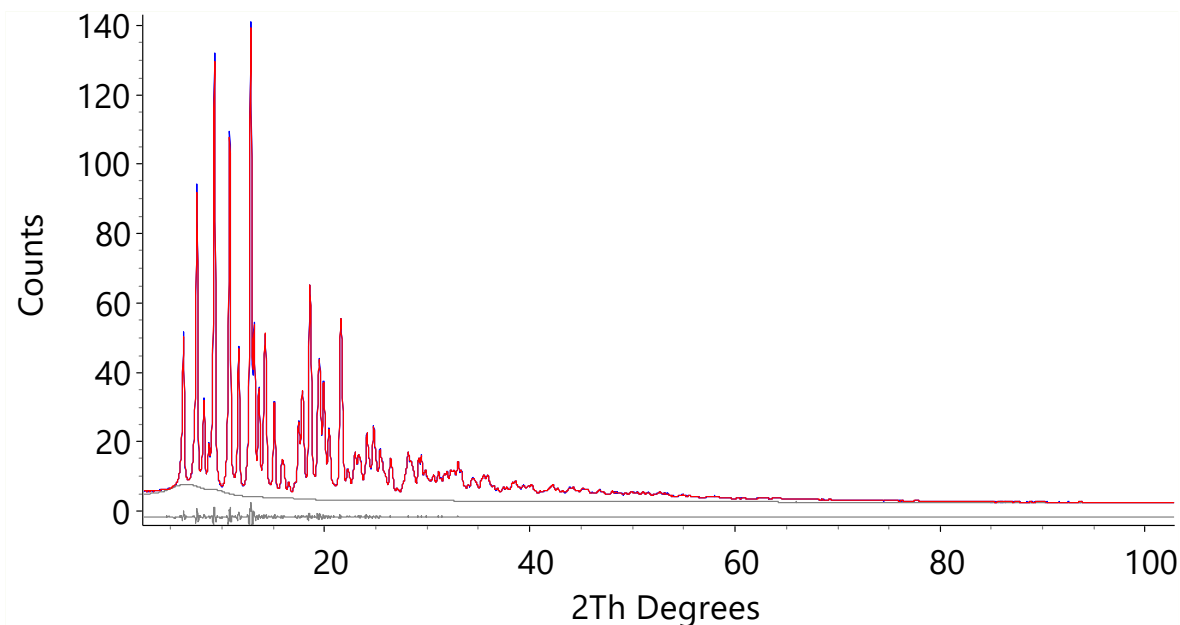
Notice the display of the background line as well as the small difference plot. When rerunning, *operation*=0, the peaks at the *xo_ls* phase is not recreated if they are already present. It may be necessary, therefore, to delete the peaks at the *xo_ls* phase when rerunning *operation*=0. When *use_simple_bkg_penalty*=0, the full background penalty is used which relies on counting statistics. For data that does not obey counting statistics, the macros *Scale_Yobs_By* can be used to scale the observed diffraction pattern. This scaling is performed using the *user_y* keyword as follows:

```
user_y data_file Data_File
yobs_eqn data.sst = data_file Scale_Yobs_By;
    min = Start_X; max = Finish_X; del = Step_X;
```

Note, *user_y* can also be a function of the reserved parameter *X*. The input created for the Kernel can be viewed in TOPAS.LOG.

11.1.2 ....Operation 1 – Generation G(r) from the fitted peaks

The macro *Average_f* is used to calculate the average atomic scattering factor *<f>* for operation 1.0. For X-ray data, a rough estimate of the atomic species is helpful; for neutron data an estimate is not required. Applying smoothing functions on $F(Q)$ such as the Lorch and Soper-Lorch functions is not recommended. Instead, applying three hat convolutions directly to $G(r)$ is faster and more accurate. At operation 1.1 the ideal pattern is converted to Q space. Operation 1.2 generates $F(Q)$ by fitting a polynomial to $Ideal(Q)$ where:

$$F(Q) = Ideal(Q) - Poly\_FQ$$

fq_poly describes *Poly_FQ* using the Chebyshev polynomial of *bkg*; the optimum number of coefficients is difficult to determine automatically. Its best to inspect the plots produced by operation 1; these are generated and loaded into the GUI and, using the Tiling option, looks like:
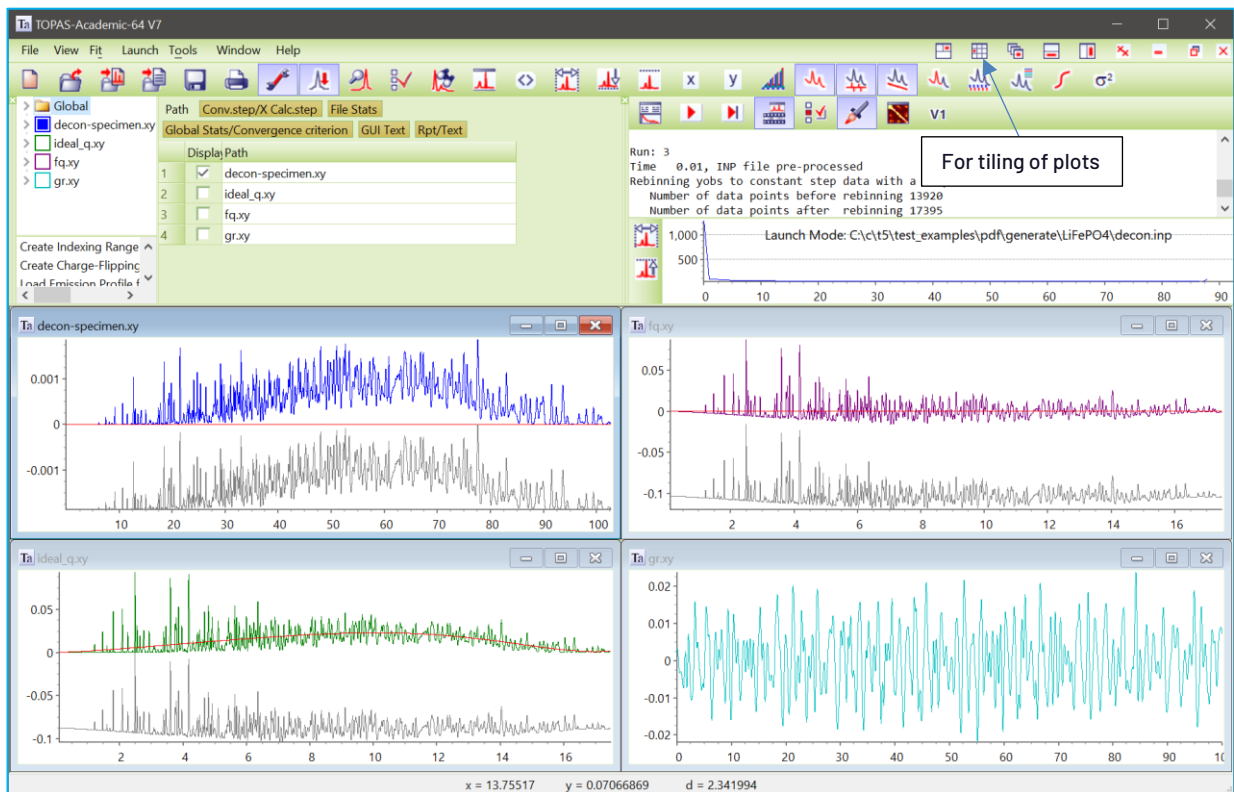
Changing *fq_poly* and rerunning operation 1 updates the four plots; this updating is achieved using the keyword *gui_reload*. Using the structure of LiFePO4, the generated $G(r)$ can be fitted-to by setting *operation*=2. With *use_narrow_peak_shape*=0 we get:
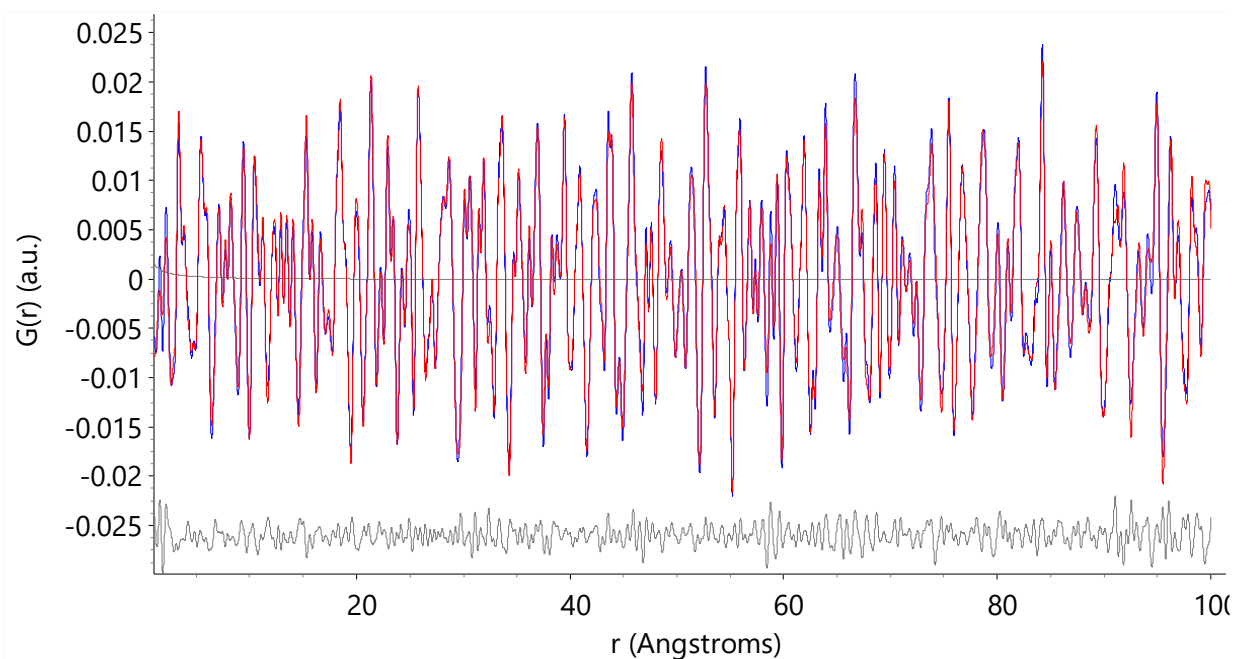
The grey line at the center of the plot is a correction added to the calculated $G(r)$ using:

```
fit_obj = a1 Cos(a2 X + a3) / X;
```

If this grey line is significant in intensity, then the value of $F(Q=0)$ is incorrect. Controlling the behaviour of $F(Q)$ at the start and end of the Q range can be done from the *FQ_Bkg_Penalty* macro. For example, $F(Q=0)=0$ can be set using following penalty:

```
      penalty = Bkg_at(X1)^2;
```

For *operation* 1; intermediate pre-processed text fed to the kernel can be sent to TOPAS.LOG (or TC.LOG) for viewing by setting *suspend_writing_to_log_file* to 0. For the current example, TOPAS.LOG for the operation 1.0 part is as follows (comments added):

```
iters 0
yobs_eqn aac.sst = 1; min 0.01 max = 103; del 0.0025
   gui_ignore
   Out_XDD_SST(decon.sst) ' Not expanded for clarity
      ' Output Ycalc / (polarization * <f>)
      = Ycalc / (( (1 + Cos(X 3.14159265358979/ 180)^2) 1 / (Sin(X
      3.14159265358979/360)^2 Cos(X 3.14159265358979/360))) (f0__(
      0.974637,0.158472,0.811855,0.262416,0.790108,0.002542,4.334946,0.342451,97.10296
      6,201.363831,1.409234 ) + f0__( 12.311098,1.876623,3.066177,2.070451,6.975185,-
      0.304931,5.009415,0.014461,18.743040,82.767876,0.346506 ) + f0__(
      1.950541,4.146930,1.494560,1.522042,5.729711,0.155233,0.908139,27.044952,0.07128
      0,67.520187,1.981173 ) + 4 f0__(
      2.960427,2.508818,0.637853,0.722838,1.142756,0.027014,14.182259,5.936858,0.11272
      6,34.958481,0.390240 ))^2);
   lam ymin_on_ymax 0.0005 la 1 lo 0.5609 lg 1e-6
   th2_offset = (-0.00730929318);
   gauss_fwhm 0.05 ' Use narrow deconvoluted peak
   xo_Is
      extra_X_left = Max(X1 - Max(X1 - 1, 0.1), 0);
      extra_X_right = Max(Min(X2 + 1, 179.9) - X2, 0);
      fn dfn (x, a0, a1) = (a0 - a1)^2 / Max(a0 + a1, 1e-6);
      default_I_attributes 1e-6 min 0 val_on_continue = Val Rand(0.5, 2) + 1e-4;
      create_pks_fn dfn create_pks_name $ a
      xo 1.40009871 I a50_ 0.0178524321`
      xo 1.42009871 I a50_ 0.0178524321`
      xo 1.44009871 I a50_ 0.0178524321`
      …
```

The actual generation of $G(r)$ occurs when Run_Number = 3; its INP text looks like:

```
   iters 0
   xdd fq.sst
      gui_reload
      lam ymin_on_ymax 0.0005 la 1 lo 0.5609 lg 1e-6
      rebin_with_dx_of 0.001
      pdf_generate {
         dr = 0.01;
         r_max = 100;
         gr_sst_file = "gr";
         hat = 4.4934 / (17.5); num_hats = 3;
      }
```

11.1.3 ....Correcting the PDF due to a zero error in reciprocal space

A zero-error added to peak positions in reciprocal can be subtracted from the deconvoluted pattern of operation 1.0. Thus, a zero-error determined from fitting to a standard in reciprocal space needs to be subtracted from the deconvoluted pattern from within the *Deconvoluted_Peak_Shape* macro.

11.1.4 ....Generating F(Q) from G(r) - gr_to_fq

The LIFEPO4\GR-TO-FQ.INP file creates $G(r)$ from an $F(Q)$ file at Run_Number 0, then in Run_Number 1 it uses the newly created $G(r)$ to reproduce the original $F(Q)$ using *gr_to_fq*. The INP file is as follows:

```
num_runs 3
#if (Run_Number == 0)
   xdd fq-original.sst
      rebin_with_dx_of 0.005
      lam ymin_on_ymax 0.0005 la 1 lo 0.5609 lg 1e-6
      pdf_generate {
         dr = 0.01;
         r_max = 300;
         gr_sst_file = "gr-from-fq";
      }
#elseif (Run_Number == 1)
   xdd gr-from-fq.sst
      gui_ignore
      lam ymin_on_ymax 0.0005 la 1 lo 0.5609 lg 1e-6
      pdf_generate {
         dr = 0.00125;
         r_max = 17.5;
         gr_sst_file = "fq-from-gr";
         gr_to_fq 1
      }
#elseif (Run_Number == 2)
   xdd fq.sst
      rebin_with_dx_of 0.01
      user_y fq_from_gr fq-from-gr.sst

      prm a 1 min 1e-6
      fit_obj = fq_from_gr a;
   #endif
```

Run_Number 3 fits the newly created $F(Q)$ to the original $F(Q)$; the result showing the reproduced $F(Q)$ (in red) and the original $F(Q)$ (in blue) has a small difference plot and is as follows:
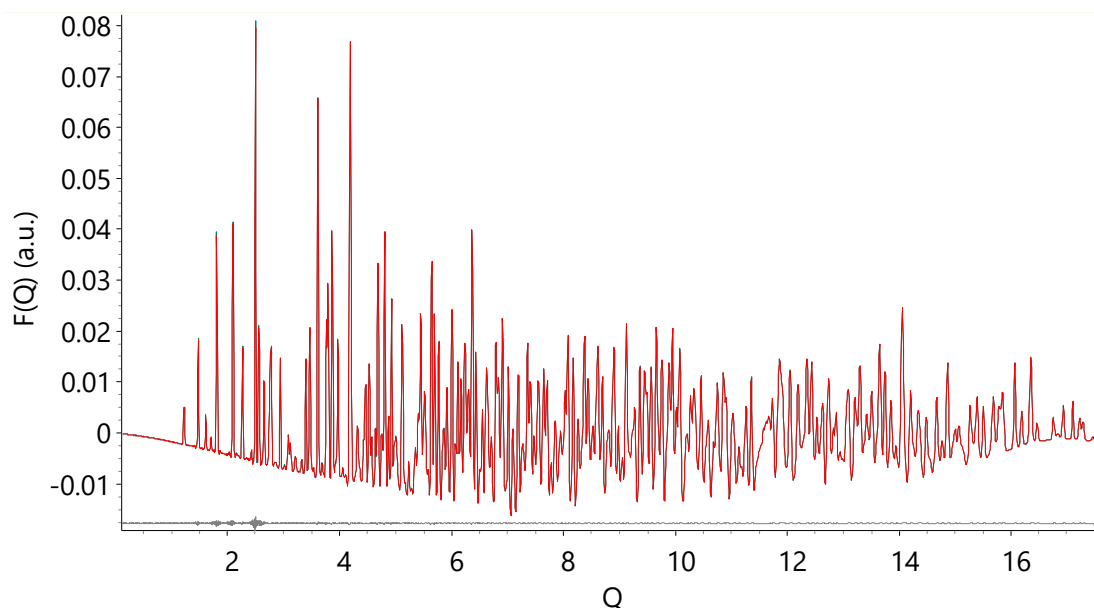
11.1.5 ....PDF-Generating - Fullerene

In this example *G*(*r*) from TOPAS is compared to *G*(*r*) from GudrunX for Fullerene. The INP file is:

```
Include_PDF_Generate
'----------------------------------------------------
'               START USER INPUT SECTION
'----------------------------------------------------
macro Data_File        { i15-1-20401_tth_det2_0.xy }
macro Capillary_Scan   { i15-1-20398_tth_det2_0.xy}
macro Capillary_Rebin  { 0 } ' Smooth the capillary scan. Zero means no smoothing
#prm operation = 1; ' Set to 0 to fit to reciprocal space data
                    ' Set to 1 generate F(Q) and G(r)
                    ' Set to 2 to fit structure to G(r)


#prm use_narrow_peak_shape  = 1; ' Use narrow peak shapes in the generating G(r)
'----------------------------------------------------
' Inputs for reciprocal space fit, operation == 0
#prm lab_no_monochromator   = 0; ' Set to 1 if using Laboratory instrument
#prm use_Xo_Is_phase        = 0; ' Set to 0 if not fitting peaks
#prm use_bkg_penalty        = 1;
#prm use_simple_bkg_penalty = 1; ' Set to 1 if counting statistics is not right
                                 ' or maybe when there's Fluorescence
macro & Bkg_Weighting              { 1 }
macro & Intensity_Penalty_Weighting { 1 }
macro & Scale_Peaks   { 1 } ' Useful if capillary absorption is inhibiting fitting.
macro & Scale_Yobs_By { 1 } ' Useful if data does not obey counting statistics.

prm pc0  1.09673044`     ' Multiplies Capillary by (pc0 + pc1 x0)
prm pc1  0.146927936     ' Comment out if not using Capillary as background.
inp_text fluorescence_bkg { }
inp_text fit_extra
   {
      penalty = 10000 (Bkg_at(X2) + (pc0 + pc1) Value_at_X(cap_, X2) - Yobs_at(X2))^2;
   }
macro Start_X      { 0.6   }
macro Finish_X     { 59.9  }
macro Step_X       { 0.02 } ' Set to 0 to use measured step size.
                            ' Set to non-zero if scale_yobs_by is use.
                            ' Set to non-zero if unequal x-axis.
'----------------------------------------------------
' Input for generating F(Q) - operation == 1
macro & Average_f            { f0_C }
macro & Qmin                 { 0.5  }
macro & Qmax                 { 25   }
macro & Soper_Lorch_Constant { 1.1 } ' Used for comparison purposes
macro & Exp_Constant         { 0 }
macro & Lorch_Constant       { 0 }
inp_text fq_poly
   {
      bkg @  0 0 0 0 0 0 0 0 0
   }
macro FQ_Bkg_Penalty  { }
'----------------------------------------------------
' Inputs for generating G(r) from F(Q), operation == 1
macro R_Max        { 50 }
macro dR           { 0.01 }
```

```
macro Num_Hats      { 0 } ' Best smoothing funcion for speed and accuracy
macro & Hat_Size    { 4.4934 / Qmax }
'-------------------------------------------------
' Reciprocal space peak details, operation == 0
macro Full_Emmision_Profile
   {
      lam ymin_on_ymax 0.0005 la 1 lo 0.161669 lg 1e-6
   }
macro Deconvoluted_Emmision_Profile
   {
      Full_Emmision_Profile
   }
macro Full_Peak_Shape_Specimen { }
macro Full_Peak_Shape { }
macro Deconvoluted_Peak_Shape
   {
      Deconvoluted_Emmision_Profile
      #if (use_Xo_Is_phase == 0)
         ' Using (Yobs - background); ie. no peak shape
      #elseif (use_narrow_peak_shape)
         ' Use Narrow peak shape
         gauss_fwhm 0.05
      #else
         ' Use Full peak shape
         Full_Peak_Shape_Specimen
      #endif
   }
macro & LP_Factor_
   {
      #if (lab_no_monochromator) (1 + Cos(X Pi/ 180)^2) #endif
      1 / (Sin(X Pi/360)^2 Cos(X Pi/360)) ' Lorentz factor
   }
'-------------------------------------------------
'              END USER INPUT SECTION
'-------------------------------------------------
Include_PDF_Generate_Common
'-------------------------------------------------
```

In this example peaks are not fitted and as such *use_Xo_ls_phase*=0, *use_simple_bkg_penalty*=1. *fluorescence_bkg* is left empty as Fluorescence is not present. *fit_extra* is used where a *penalty* is applied equating the bkg_tot to the Yobs value at the end of the diffraction pattern. Note, the use of the Value_at_X function. bkg_tot in this example comprises a *fit_obj* which corresponds to (pc0 + pc1 x)**Capillary*. In this example the *Soper_Lorch_Constant* was used in order to match Gudrun X. $G(r)$ generated for TOPAS (in red) and GudrunX (in Blue) is as follows:

12. PDF REFINEMENT

[*xdd*]...	Examples
[*pdf_data*]	
[*scale_phase_X*[1] E]...	**INP files**
[*fit_obj*[1] E]...	TEST_EXAMPLES\PDF\
[*start_X* #][*finish_X* #]	BEQ-2.INP
[*rebin_with_dx_of*[1] !E]	BEQ-2-CREATE.INP
[*rebin_start_x_at* !E]	BEQ-3.INP
[*weighting* !E]	BEQ-3-CREATE.INP
[Tpdf_convolute]...	PDF-1.INP
[*str*]...	PDF-2.INP
[*scale_phase_X1* E]...	ALVO4\
[*scale* E]	STRUCTURE-SOLUTION-CREATE.INP
[*view_structure*]	STRUCTURE-SOLUTION.INP
[*rigid*]...	RIGID.INP
[*occ_merge* $sites]...	OCC-MERGE-PBSO4\
[*pdf_scale_simple*]	CREATE.INP
[*pdf_zero*[1] E]	OCC-MERGE-TEST.INP
[*pdf_ymin_on_ymax* 0.001]	OCC-MERGE.INP
[*pdf_info*]	**Data files**
[Tpdf_convolute]...	TEST_EXAMPLES\PDF\
[*pdf_for_pairs* $sites_1 $sites_2]...	BEQ-2.XY
[*pdf_only_eq_0*]	BEQ-3.XY
[*pdf_gauss_fwhm*[1] E]	ALVO4\ALVO4.XY
[Tpdf_convolute]...	OCC-MERGE-PBSO4\PBSO4.XY
Tpdf_convolute	
[*pdf_convolute*[1] E]...	
[*min_X* !E]	
[*max_X* !E]	
[*convolute_X_recal* !E]	

[1]) Can be a function of the reserved parameter name *X*; *X* corresponds to r for PDF data.

PDF refinement as implemented operates at speed (Coelho, 2015). PDF patterns are treated as an *xdd* where most *xdd* keywords can be used. PDF patterns can be refined simultaneously with other types of *xdd* patterns where the latter can comprise x-ray dependent or x-ray independent phases. Penalties, restraints and keywords such as *rigid*, *atomic_interaction*, *sites_geometry*, *sites_distance* etc. can all be used. *pdf_data* tells the program that the data set is of $G(r)$ type. Let's write $G(r)$ as:

$$G(r) = s_1 S(r) / r - s_2 r$$

where r corresponds to the x-axis, s_1 and s_2 are constants and $S(r)$ are the pairs. *pdf_scale_simple* tells the program to calculate $S(r)/(N_p r)$ only. *pdf_ymin_on_ymax* defines the min/max value for the PDF Gaussians in-regards-to the x-axis extents of the Gaussians; the default value of 0.001 in

typically sufficient. *pdf_for_pairs* can be used to select site pairs using the site name; for example:

```
        pdf_for_pairs "V* Al* !O2" *
```

The '!' character excludes the sequence from the wild card string, see section 17.20. Multiple *pdf_for_pairs* can be defined. *pdf_only_eq_0* informs the parent *pdf_for_pairs* that only equivalent position 0 is to be considered. *pdf_gauss_fwhm* is used to write the width equation for the pairs selected by *pdf_for_pairs*. If all pairs are described by *pdf_for_pairs* then the associated *beq*'s are not used; the user is informed of unused *beq*'s. Consider the following abbreviated INP segment:

```
        site Al1 ... beq 1
        site O1  ... beq 1
        pdf_for_pairs Al1 Al1 pdf_only_eq_0 pdf_gauss_fwhm 0.1 ' Line A
        pdf_for_pairs Al1 O1  pdf_only_eq_0 pdf_gauss_fwhm 0.2 ' Line B
        pdf_for_pairs Al1 O1                pdf_gauss_fwhm 0.3 ' Line C
```

The FWHMs of the interactions are as follows:

Al1-O1 : Interactions for equivalent-position-0 described using Line B.

Al1-O1 : Interactions excluding equivalent-position-0 described using Line C.

O1-O1 : Interactions described using beq's.

pdf_info displays the interactions in matrix form; for the above INP segment we have:

```
        pdf_info
        {
          - = No pdf_for_pairs defined hence beq's used
          0 = pdf_for_pairs defined with pdf_only_eq_0
          1 = pdf_for_pairs defined without pdf_only_eq_0
          2 = two pdf_for_pairs defined, one with and one without pdf_only_eq_0

            Al1     -2
            O1      2-
        }
```

The matrix is in purple. *pdf_for_pairs* together with *beq* defaults offer great flexibility in describing peak widths. See PDF-1.INP, PDF-2.INP, BEQ-3.INP. *scale_phase_X* can be used to describe Gaussian dampening, for example:

```
        prm damp_fwhm 50 min 1e-6 max 200
        prm damp = Gauss(0, damp_fwhm);
        scale_phase_X = damp;
```

12.1 ... pdf_only_eq_0

Consider the space group *P*-1 with two equivalent positions, E0 and E1:

```
        E0)    x, y, z
        E1) -x, -y, -z
```

The PDF comprises interactions between all atom pairs. From symmetry, only interactions between E0 and the rest of the atoms are calculated. For a two-atom structure in *P-1*, with atoms A and B, the PDF comprises unique interactions between the following pairs:

A0-A0, A0-A1, A0-B0, A0-B1, B0-B0, B0-B1, B0-A1

Each interaction can be defined separately using a combination of *beq* and *pdf_for_pairs*. If the following is defined:

```
site A beq = a;
site B beq = b;
pdf_for_pairs A B              pdf_gauss_fwhm = ab;
pdf_for_pairs B B pdf_only_eq_0   pdf_gauss_fwhm = b0b0;
```

then the 7 types of interactions would have broadening as follows:

Pair	Gauss FWHM
A0-A0	Sqrt(a 2 Ln(2) / Pi^2)
A0-A1	Sqrt(a 2 Ln(2) / Pi^2)
A0-B0	ab
A0-B1	ab
B0-B0	b0b0
B0-B1	Sqrt(b 2 Ln(2) / Pi^2)
B0-A1	ab

For equivalent-position-0 and for distances within 10 Å then the following is required:

```
pdf_for_pairs * * pdf_only_eq_0
   pdf_gauss_fwhm = If(X < 10, something, 0);
```

pdf_info can be useful for seeing what is being used. Consider three sites A, B, C. For three sites there are (N^2+N)/2=6 types of atom-atom interactions:

A-A, A-B, A-C, B-B, B-C, C-C

Each of these can have broadening defined in three different ways, take A-A for example:

```
1) site A ... beq
2) pdf_for_pairs A A
3) pdf_for_pairs A A pdf_only_eq_0
```

Use of *pdf_only_eq_0* results in three types of A-A interactions:

i)   interaction where none are equivalent position zero.

ii)   interaction where both are equivalent position zero.

iii) interaction where one is equivalent position zero and the other not.

If case (2) is used then (i), (ii) and (iii) all use case (2); for example:

```
site A ... beq ... pdf_for_pairs A A ...
```

If case (3) is used, then *beq* is used for (i) and (iii) and case (3) is used for (ii); for example:

```
site A ... beq ... pdf_for_pairs A A pdf_only_eq_0...
```

If both case (2) and (3) are used then *beq* is ignored and case (2) is used (i) and (iii), and case (3) for (ii); for example:

```
site A ... beq ...
    pdf_for_pairs A A...                ' (i) and (iii)
    pdf_for_pairs A A pdf_only_eq_0... ' (ii)
```

12.2 ... Inter and Intra molecule FWHMs

pdf_for_pairs can be used to assign different interaction types between molecules. For example, to set the bond lengths for the atom Al1 of AlVO$_4$ (see PDF-2.INP) for equivalent-position-0 only, the following could be used:

```
prm intra_molec 0.01 min 1e-6
pdf_for_pairs Al1 "O1 O2 O3 O4 O5 O6" pdf_only_eq_0
    pdf_gauss_fwhm = intra_molec;
```

The calculated pattern from PDF-2.INP therefore becomes:

Calculated G(r) with a molecule with narrow FWHMs
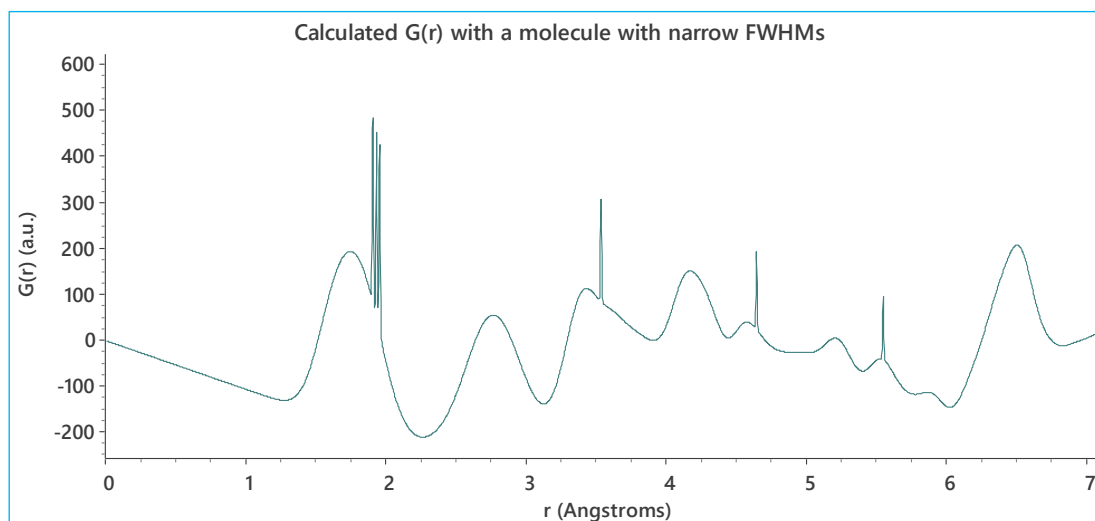
Notice the 6 spikes; they correspond to the Al1 bonds with narrow FWHMs. If we wanted Al1 bonds that are not equivalent-position-0 to be different to the *beq*'s then we could use:

```
prm inter_molec 0.1  min 1e-6
prm intra_molec 0.01 min 1e-6
pdf_for_pairs Al1 "O1 O2 O3 O4 O5 O6" pdf_only_eq_0
    pdf_gauss_fwhm = intra_molec;
pdf_for_pairs Al1 "O1 O2 O3 O4 O5 O6"
    pdf_gauss_fwhm = inter_molec;
```

This gives the following calculated pattern where we see the various Al1 bonds.

Calculated G(r) with a molecule with narrow FWHMs
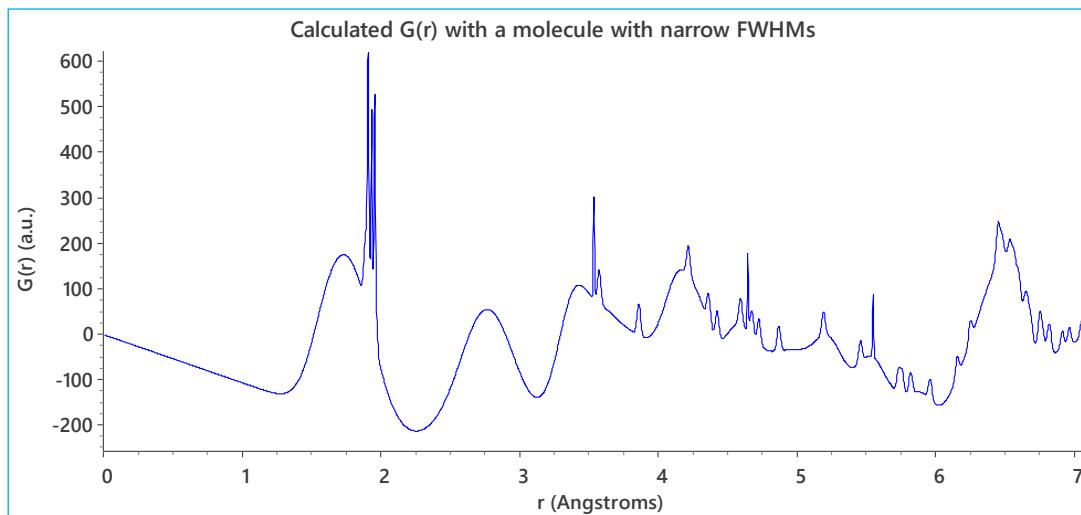
The corresponding output from *pdf_info* becomes:

```
pdf_info
{
- = No pdf_for_pairs defined
0 = pdf_for_pairs defined with pdf_only_eq_0
1 = pdf_for_pairs defined without pdf_only_eq_0
2 = two pdf_for_pairs defined, one with pdf_only_eq_0 and one without pdf_only_eq_0

    Al1     ------222222------
    Al2     -----------------
    Al3     -----------------
    V1      -----------------
    V2      -----------------
    V3      -----------------
    O1      2----------------
    O2      2----------------
    O3      2----------------
    O4      2----------------
    O5      2----------------
    O6      2----------------
    O7      -----------------
    O8      -----------------
    O9      -----------------
    O10     -----------------
    O11     -----------------
    O12     -----------------
}
```

An exception is thrown if the same interaction is referenced in more than one *pdf_for_pairs*, for example, the following will throw an exception as Al1–O1 is referenced twice:

```
        pdf_for_pairs Al1 "O1 O2 O3 O4 O5 O6" pdf_only_eq_0 ...
        pdf_for_pairs Al1 O1 pdf_only_eq_0 ...
```

The following will not throw an exception:

```
        pdf_for_pairs Al1 "O1 O2 O3 O4 O5 O6" pdf_only_eq_0 ...
        pdf_for_pairs Al1 O1 ...
```

12.3 ... Instrument Sinc function sinc-1.inp

In SINC-1.INP, *pdf_convolute* is used at the *xdd* level to convolute a Sinc function into phases:

```
pdf_convolute = Sin(Qmax X+q3)/If(Abs(X) < 0.5 Step_Size, If(X < 0,-q2,q2),X);
    min_X = -conv_max;
    max_X =  conv_max;
```

SINC-1.INP also uses an *xo_ls* phase defined as:

```
xo_Is
    NoThDependence(0.0001)
    xo 10 I @ 100
    peak_type pv
        pv_lor 0.5
        pv_fwhm 2
```

pdf_convolute operates on PDF type phases only; the *xo_ls* phase is untouched. Note the phase dependent use of an emission profile as defined in the *NoThDependence* macro. Multiple *pdf_convolute*'s can be described at the global, *xdd*, *str* and *pdf_for_pairs* levels. Use of *pdf_convolute* as a dependent of *pdf_for_pairs* is slower than at the other levels; thus where possible use *pdf_convolute* outside of *pdf_for_pairs*.

12.4 ... Weighting of PDF and 2-Theta type data

PDF and 2θ data can be of very different intensities; *xdd_sum* can be used to modifying the weighing of these data in order to give the patterns approximately similar weights. For example:

```
xdd file1.xy
    xdd_sum !sum1 = Abs(Yobs);
    weighting = 1 / sum1;
xdd file2.xy
    xdd_sum !sum2 = Abs(Yobs);
    weighting = 1 / sum2;
    pdf_data
```

12.5 ... Test_examples\pdf\beq-2.inp

BEQ-2-CREATE.INP generates a simulated pattern for BEQ-2.INP which:

- comprises the structure of $AlVO_4$,
- 3 types of *beq* parameters,
- *beq* is a function of X (i.e. $X = r$) and hence peak widths are a function of X,
- demonstrates the use of *pdf_zero*,
- demonstrates the use of *rebin_with_dx_of* and *rebin_start_x_at*.

12.6 ... Test_examples\pdf\beq-3.inp

BEQ-3-CREATE.INP generates a simulated pattern for BEQ-3.INP; it demonstrates the use of *pdf_for_pairs*.

12.7 ... Speeding up refinement with rebin_with_dx_of

Increasing the x-axis step size of PDF data can speed up refinement; see BEQ-2.INP. The step size must be of equal size and the start of the x-axis needs to be an integral multiple of the step size. Data can therefore be rebinned to increase step size as follows:

```
macro Rebin_Step   { 0.015 }
rebin_with_dx_of Rebin_Step rebin_start_x_at Rebin_Step
```

Rebinning is akin to collecting the data at a larger step size. All data is included; counts after rebinning is equal to counts before rebinning. *esd*'s associated with the data are also rebinned. *rebin_start_x_at* can be used to place the start of the data at an integral multiple of the step size. In BEQ-2.INP parameters such as scale are written in terms of the rebin step size to reflect the fact that the scaling of the data is changed due to rebinning.

12.8 ... Refining on beq parameters

Modify the *BB* macro so that its empty as in the following:

```
macro BB {  } ' Enter ! to not refine, beq including low angle fwhm sharpening
```

This results in refinement of four independent *beq* parameters including the low angle sharpening parameter of erf_a as seen in the following:

```
macro Beq(c, v)
    {
        #m_argu c
        If_Prm_Eqn_Rpt(c, v, min 1e-6 max 10 val_on_continue = Rand(.1, 2); )
        beq = CeV(c, v) Erf_Approx( erf_a X);
    }
```

The Rwp plot is:
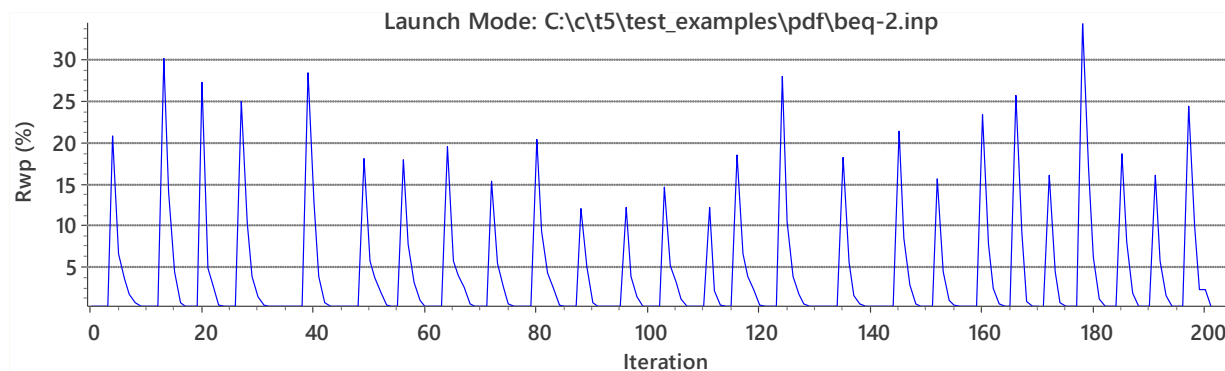
This type of convergence is indicative of correct derivative calculation. Convergence for coordinates, occupancies, lattice parameters and *pdf_zero* are similar.

12.9 ... Structure Solution, Simulated Annealing

PDF\ALVO4\STRUCTURE-SOLUTION-CREATE.INP creates a simulated pattern for STRUCTURE-SO-LUTION.INP. It's a simulated annealing refinement with all coordinates starting at zero and with anti-bump penalties applied using:

```
AI_Anti_Bump(O* , O* , 2.4, 1, 5)
AI_Anti_Bump(Al*, O* , 1.6, 1, 5)
AI_Anti_Bump(Al*, Al*, 2.8, 1, 5)
```

The correct solution is found as seen in the following:
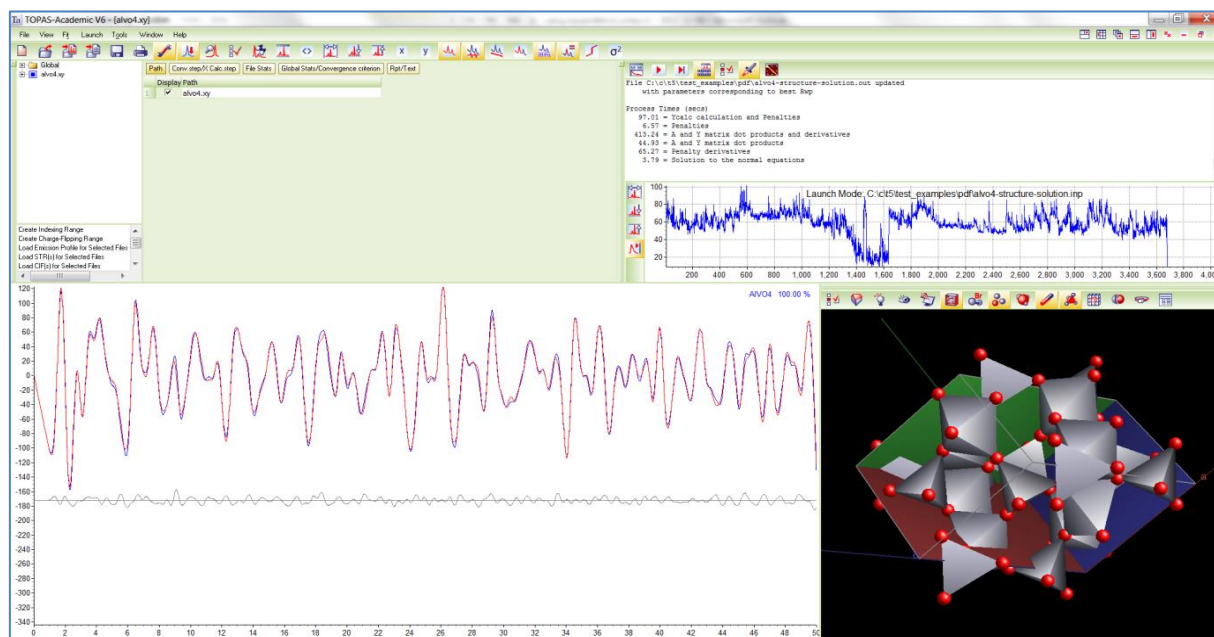
The range of convergence for atomic coordinates are smaller than with reciprocal space as in normal Rietveld refinement. This is because the coordinates, in the PDF case, changes peak positions rather that peak intensities; with the former having a narrow range of convergence. It may be possible to increase the range of convergence for the PDF case by increasing the peak widths; this however comes at the expense of resolution and it may result in an even smaller range of convergence.

12.10 . Rigid bodies with PDF data

PDF\ALVO4\RIGID.INP operates on simulated data created by STRUCTURE-SOLUTION-CREATE.INP. It demonstrates the use of rigid bodies with PDF data.

12.11 . Occupancy merging with PDF data

PDF\OCC-MERGE-PBSO4\OCC-MERGE.INP operates on simulated data created by CREATE.INP. It demonstrates the use of *occ_merge* with PDF data.

12.12 . Equivalence of pdf_gauss_fwhm and beq for one atom type

PDF\SI1.INP comprises an option to use *beq* or *pdf_gauss_fwhm*. For the beq case we have:

```
        beq = width;
```

and for *pdf_gauss_fwhm* we have:

```
        pdf_gauss_fwhm = Sqrt(width 2 Ln(2) / Pi^2);
```

The above cases are equivalent when all atoms are of the same type.

13. STACKING FAULTS

[*site* $name]...
    [*layer* $layer]
[*stack* $layer]...
    [*sx* E][*sy* E][*sz* E]
    [*generate_these* $sites]
        [*generate_name_append* $append_to_site_name]

The super cell approach to stacking faults has been implemented. *layer* identifies a site as belonging to a layer called $layer; *stack* applies a stacking vector { *sx*, *sy*, *sz* } to the named layer. Structures factors are generated in the usual manner with a shift applied corresponding to the stacking vector. *stack* operates in any space group. Sites that do not belong to a layer are treated as un-stacked and their structure factors are generated in the usual manner. *generate_these* generates the sites found in $sites for the *stack* with coordinates that reflect original $sites positions plus the stacking vector. *generate_name_append* appends $append_to_site_name to the generated site. The generated sites have occupancies set to zero which signals a dummy site. Dummy sites do not take part in structure factor calculations and hence speed is not hindered. The dummy sites allow for graphical display of the layers; i.e.
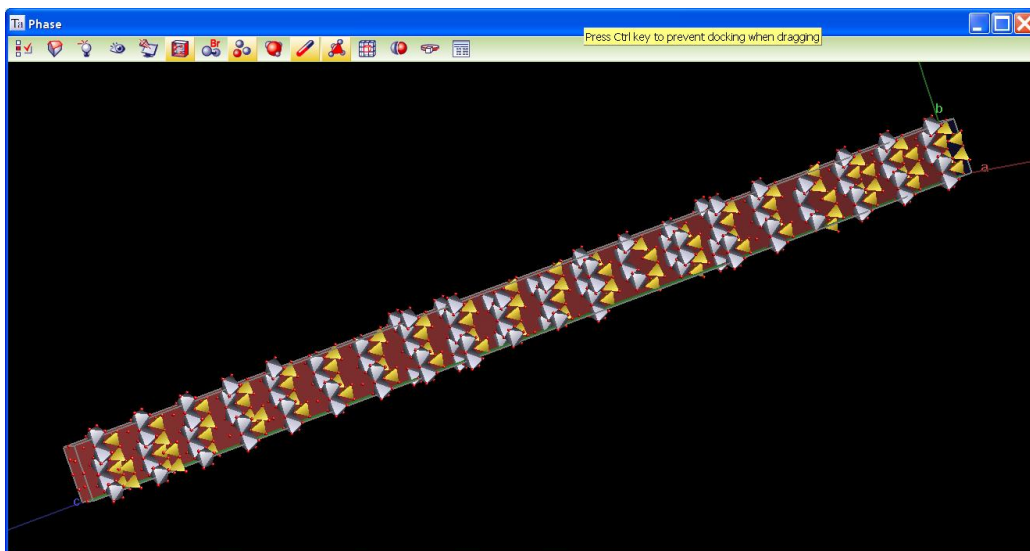
Importantly penalties can operate on dummy sites which allow for restraints such as *Distance_Restrain*. The following rules govern the behavior of sites marked with *layer*:

- A site marked with *layer* cannot take part in restraints.
- A site marked with *layer* is not displayed graphically.
- A site generated using *generate_these* can take part in restraints.
- A site not marked with *layer* can take part in restraints.

For example:

```
space_group P1
site 01 ... layer A
site 02 ... layer A
```

```
stack A
    sx ...
    generate_these 01
        generate_name_append _1
append_fractional
    in_str_format
```

will output for *append_fractional* the following:

```
site 01 ...
site 02 ...
site 01_1 ... occ 0 0
```

The TEST_EXAMPLES\STACKING-FAULTS\KAOLINITE.INP shows how to simplify the setting up of layers with the use of simple macros. Speed of calculation for structure factors are very fast and the derivatives of the stacking vectors { *sx*, *sy*, *sz* } are very fast. The main bottle neck in speed is summing the peaks to *Ycalc*. The switch "*#define* Speed" in KAOLINITE.INP shows keywords that can speed things up in the early stages of determining the stacking vectors.

13.1 ... Fitting to a Debye-formulae pattern using 'stack'

A test pattern was generated using the Debye scattering equation. The structure comprised a single atom in an Orthorhombic unit cell with 40 layers (40x40x40 unit cells) in the a-b plane shifted according to {Round(Rand(0,2))/3, Round(Rand(0,2))/3, 0}. The blue line in the following is the generated pattern comprising the average of 30 runs of the Debye scattering equation. The red line corresponds to a Rietveld fit of 6 super cell structures (1x1x40) showing that the super cell approach is a good approximation to the Debye formulae for this example.
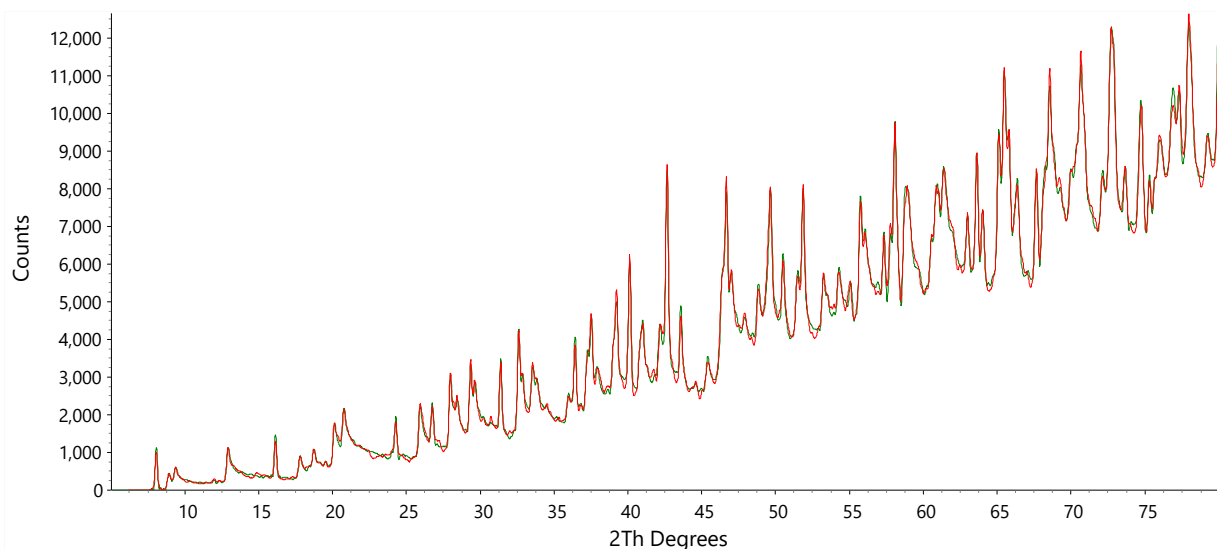
The example STACKING-FAULTS\DEBYE-NEW.INP corresponds to the Rietveld fit using the *layer* and *stack* keywords. The DEBYE-OLD.INP file corresponds to the same Rietveld fit but without the *layer* and *stack* keywords; instead layers are explicitly defined using *site* in an enlarged unit cell.

There are two time-consuming bottle necks dealt with:

1) Summing peaks to *Ycalc*

2)   Calculating structure factors for the stacked layers

The phase dependent [*del_approx* #] groups peaks from the peaks buffer whilst summing peaks to *Ycalc*; the peaks are grouped such that their 2Th positions all lie within:

$$(-del\_approx \; Peak\_Calculation\_Step) < 2Th < (del\_approx \; Peak\_Calculation\_Step)$$

Once the group is found then only the two peaks with the smallest and largest 2Th is kept. The in-between peaks have their intensities appropriated to the kept peaks. The peak buffer stretching routines have also been optimized for both accuracy and speed. The following points should be noted when working with large super cells

- The *layer* and *stack* keywords increase computational speed and reduce memory usage.
- *del_approx* increase computation speed at a relatively small cost to accuracy; a value between 1 and 3, dependent on *Peak_Calculation_Step*, is typically acceptable.
- The graphical display of 10s of 1000s of hkl ticks (there's 51584 hkls in each phase of the DE-BYE-NEW.INP) is time consuming; turning the graphical hkl-ticks option Off is worthwhile.

13.2 ... Fitting to Kaolinite data

STACKING-FAULTS\KAOLINITE.INP demonstrates the application of *stack* and *layer* with the following fit:
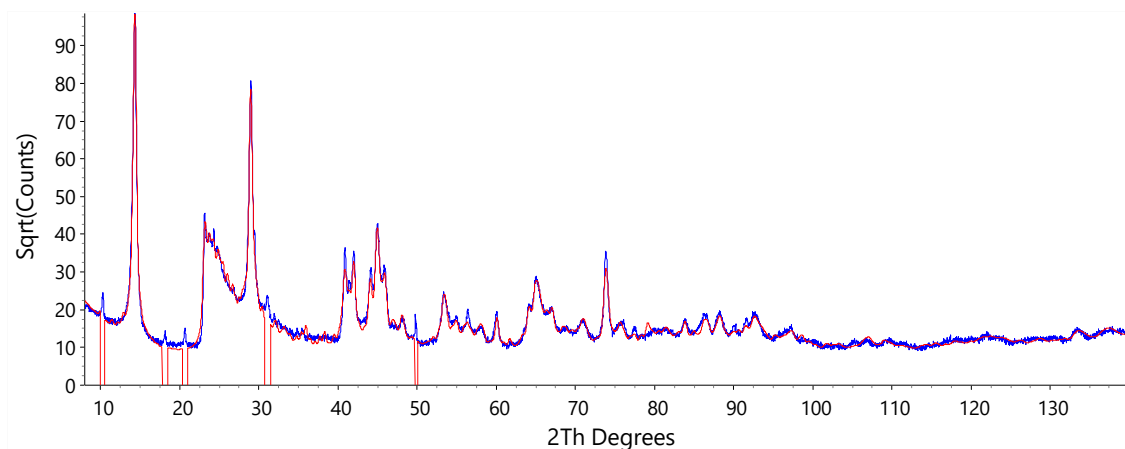
In this example the stacking vectors are refined in a simulated annealing process.

13.3 ... Stacking faults and generating sequences of layers

	Examples

[*generate_stack_sequences*]{
    [*number_of_sequences* !E]
    [*number_of_stacks_per_sequence* !E]
    [*save_sequences* $file]
    [*save_sequences_as_strs* $file]
    [*user_defined_starting_transition* $tran-
        sition_name]
    [*layers_tol* !E]
    [*n_avg* !E]
    [*num_unique_vx_vy* !N]
    [*match_transition_matrix_stats* {...}]
    [*transition* $transition_name]...
        [*use_layer* $layer]
        [*height* E]
        [*n* !N]
        [*to* $to_transition_name !E]...
            [*ta* E][*tb* E][*tz* E]
            [*a_add* E][*b_add* E][*z_add* E]
}
'Get(generated_c)
	\TEST_EXAMPLES\STACKING_FAULTS\
	    FIT-1.INP
	    FIT-2.INP
	    FIT-3.INP
	    RIETVELD-GENERATE\
	        CREATE-SEQUENCES.INP
	        RIETVELD-GENERATE.INP
	        FIT-TO-RIETVELD-GENERATED.INP
	        RIETVELD-GENERATED-200-
	        2000.XY
	        STRS-200-2000.TXT

Stacking fault generation and refinement can now be performed at speeds that make routine analysis possible (Coelho *et al.*, 2016). *generate_stack_sequences* generates sequences of stacks from the transition matrix described by the *transition* keyword. The opening and closing braces of { ... } corresponds to a block where keywords local to *generate_stack_sequences* can be used. Outside of the braces the *generate_stack_sequences* can't be used. After generation of the sequences, Get(generated_c) is updated with the average thickness of the generated sequences. It can be used to set the *c* lattice parameter.

On termination of refinement, *num_unique_vx_vy* reports on the number of unique { *sx*, *sy* } stacking vector coordinates for all layer types. *transition* defines a 'from' transition with the name $transition_name. The transition uses the layer defined in *use_layer*. *to* defines the to-transition. $to_transition_name must be a defined $transition_name. *n* returns the number of transitions generated for the corresponding *to* to-transition. *height*: can be used instead of *z_add* keywords. *ta*, *tb*: defines the stacking vector x and y coordinates in terms of the crystallographic **a** and **b** axes. *a_add*, *b_add*: defines the stacking vector x and y coordinates relative to the previous stacking vector in terms of the crystallographic **a** and **b** axes. *tz:* defines stacking vector z coordinate along the crystallographic **c** axis in Å. *add_z*: defines stacking vector z coordinate along the crystallographic **c** axis in Å relative to the previous stacking vector.

user_defined_starting_transition: if used, stacking begins at the *transition* with the name of $transition_name. Otherwise, stacking begins at the *transition* with the greatest probability according to the probability density matrix.

13.3.1 ....Generating the same stacking sequences each run

To generate the same set of stacking sequences each run the random number generated can be seeded with a constant seed using *seed*, for example:

```
seed #number
```

#number is a constant integer. Each #number generates its own unique set of random numbers. Generating identical sets of stacking sequences is useful when changes in R_{wp} that excludes stacking sequence variation is required.

13.3.2 ....The SF_Smooth macro

Stacking faulted calculated patterns can contain ripples when the peak shapes are small or when there's too few layers stacked. The *SF_Smooth* macro, defined in TOPAS.INC smooths out these ripples such that small supercells can approximate large supercells; this crease computation speed and reduces memory usage. All stacking fault examples use *SF_Smooth*; typical usage is:

```
SF_smooth(@, 1, 1)
```

The refined parameter adjusts the width of a Gaussian convolution that is dependent on hkls and the intensities of the reflections. The last argument *s* (the '1') can be used to adjust the tolerance of *peak_buffer_based_on* used in the *SF_Smooth* macro; the definition of the latter is:

```
peak_buffer_based_on = idl;
    peak_buffer_based_on_tol = Max(0.01 idl, Peak_Calculation_Step 0.5 s);
```

Reducing *s* increases the number of peaks in the peaks buffer and increases the accuracy of the calculated pattern. *s*=1 is typically sufficient.

13.3.3 ....Fitting to DIFFaX test diamond data

FIT-1.INP uses *generate_stack_sequences* to fit to data generated from the DIFFaX suite (Treacy, 1991); the INP segment that generates the sequences looks like:

```
generate_stack_sequences {
    number_of_sequences Nseqs 200
    number_of_stacks_per_sequence Nv 200
    num_unique_vx_vy  6
    Transition(1, lpc)
        to 1 = pa;      a_add = 2/3;  b_add = 1/3;   n !n1  349984
        to 2 = 1-pa;    a_add = 0;    b_add = 0;     n !n2  149781
    Transition(2, lpc)
        to 1 = 1-pa;    a_add = 0;    b_add = 0;     n !n3  149781
        to 2 = pa;      a_add = -2/3; b_add = -1/3;  n !n4  350254
}
```

The generated probability parameter pa can be determined using the *n* values as follows:

```
prm !pa_gen = (n1+n3)/(n1+n2+n3+n4); : 0.699974874
```

The fit to the DIFFaX data looks like:

Fit-1.INP, Fitting to DIFFaX test diamond data
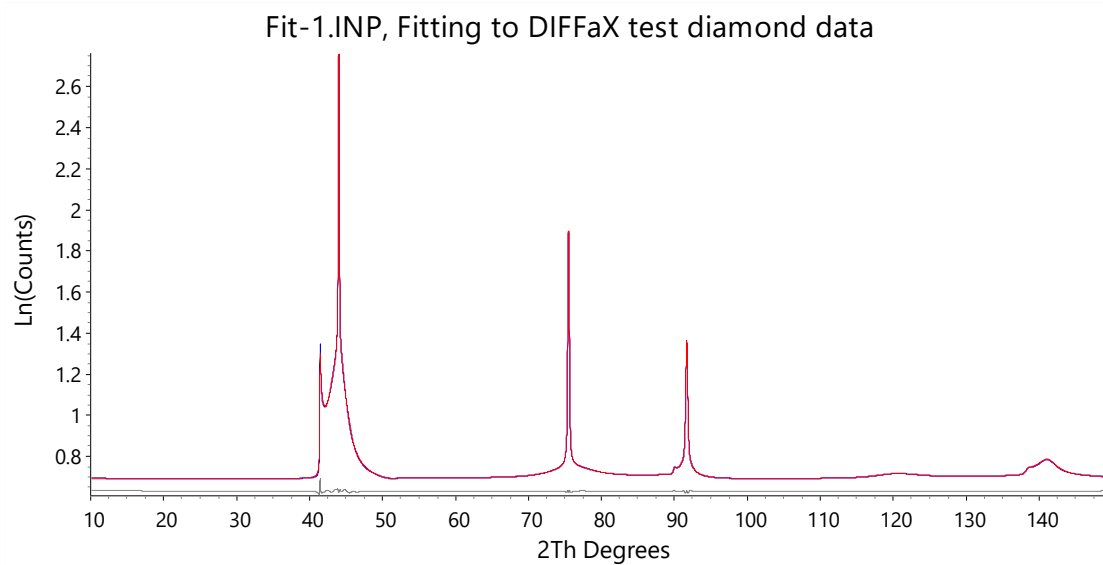
13.3.4 .... Stacking faults from layers of different layer heights

Layers of different thicknesses can be accurately modelled and with fast refinement. Here's a fit to simulated data (FIT-2.INP) for two different layer heights of 5 and 6Å.

Fit-2.INP, Fitting to test data created with different layer heights
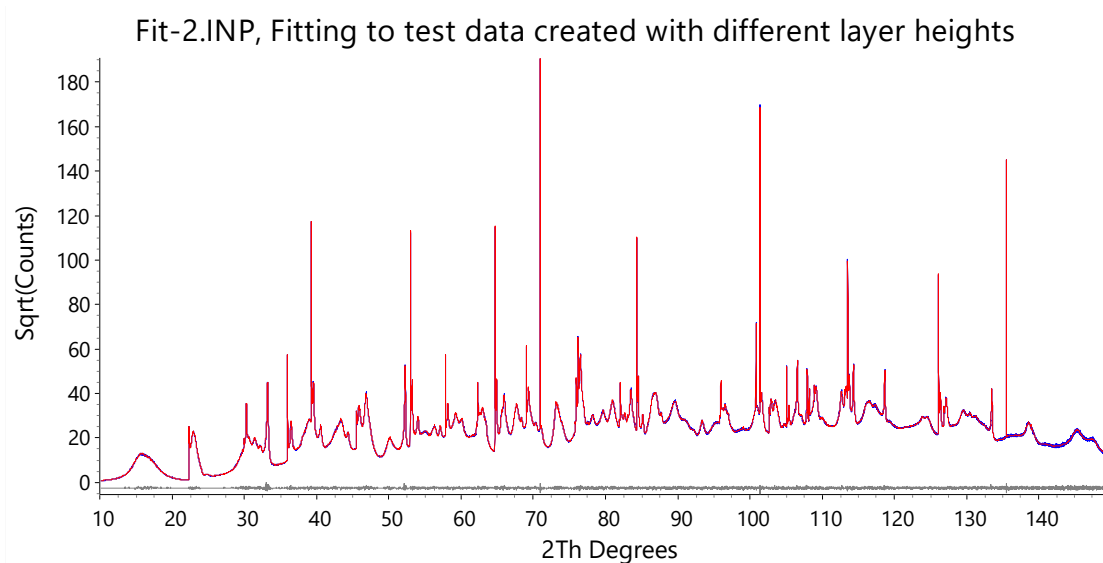
13.3.5 .... Rietveld-Generated example

The files in the RIETVELD-GENERATE directory can be used to create a stacking faulted test pattern using Rietveld refinement; the test pattern can then be refined against. CREATE-SEQUENCES.INP creates the INP format stacking sequences and places the result in the file STRS-200-2000.TXT. The file RIETVELD-GENERATE.INP can be used to create the test pattern RIETVELD-GENERATED-200-2000.XY. This test pattern can be fitted-to using FIT-TO-RIETVELD-GENERATED.INP; this INP file uses *generate_stack_sequences* and it demonstrates the accuracy and speed of the stacking fault averaging procedure. The fit to the Rietveld generated stacking faulted pattern looks like:

Fit-to-Rietveld-Generated.INP
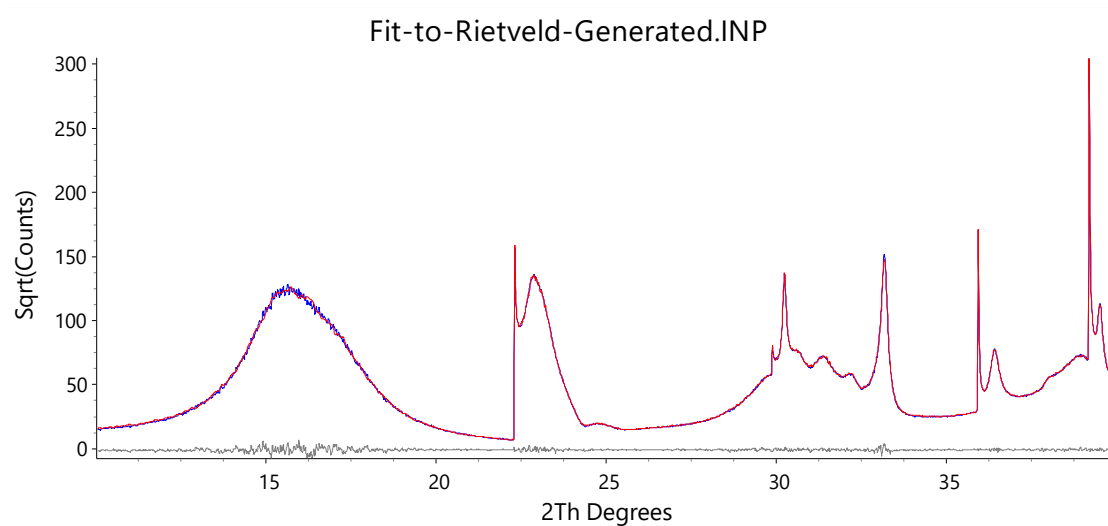
13.3.6 ....Refining on layer heights

Layer heights can be refined by refining on parameters that are a function of the *add_z* or *height* keywords. The FIT-3.INP example refines on 3 height parameters as well as the *z* fractional atomic coordinates of the sites that comprise the layers. It also lists six types of transitions which operate on three unique layer types. The transitions points to the unique layer types using *use_layer*. The *c* lattice parameter is defined and refined using the following:

```
prm qq 0 c = Get(generated_c) + 0.0001 qq; : 1828.085117
```

Get(generated_c) is also used to initialize the *z* fractional coordinates of the sites as follows:

```
prm height_Se01 7.49691
prm !zSe01 = height_Se01 / Get(generated_c);
site Se01 x 0.5 y 0 z = zSe01; occ Se 1 beq !bval 1 layer cd00
```

The fit to the test data looks like:

Fit-3.INP, Refining on stacking vector and structural parameters
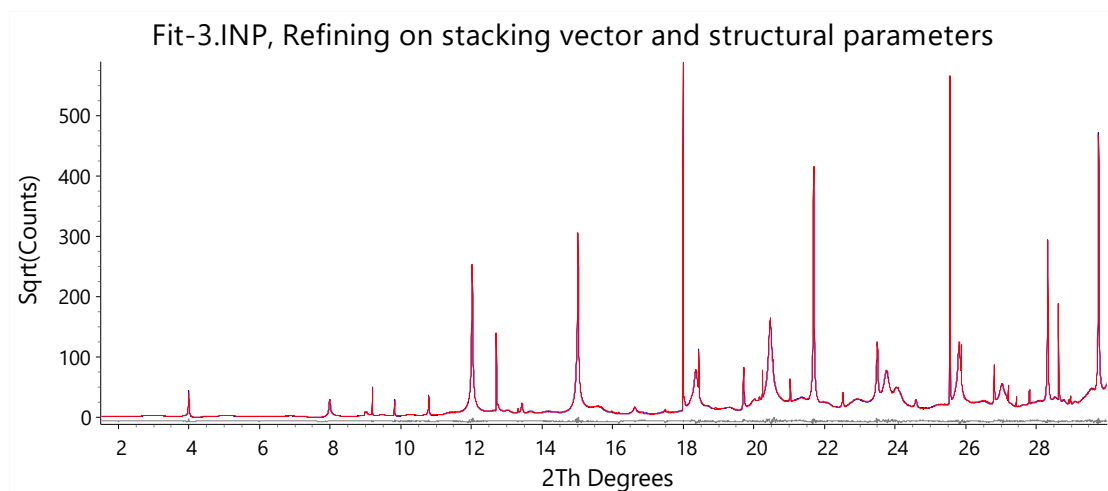
14. QUANTITATIVE ANALYSIS

[*xdd*]...
    [*mixture_MAC* #]
    [*mixture_density_g_on_cm3* #]
    [*weight_percent_amorphous* !E]
    [*elemental_composition*]
    [*element_weight_percent* $atom $Name #]...
    [*element_weight_percent_known* $atom #]...
    [*prm* = Get(sum_smvs)...]
    [*prm* = Get(mixture_MAC)...]
    [*prm* = Get(mixture_density_g_on_cm3)...]
    [*Mixture_LAC_1_on_cm*(0)]
    [*str*]...
        [*cell_mass* !E][*cell_volume* !E][*weight_percent* !E]
            [*spiked_phase_measured_weight_percent* !E][*corrected_weight_percent* !E]
    [*phase_MAC* !E]
    [*prm* = Get(sum_smvs)...]
    [*prm* = Get(smv)...]
    [*prm* = Get(sum_smvs_minus_this)...]
    [*prm* = Get_Element_Weight(atom)...]
    [*Phase_LAC_1_on_cm*(0)]
    [*Phase_Density_g_on_cm3*(0)]

Examples in directory TEST_EXAMPLES\QUANT\

The QUANT implementation to a large extent is written internally using the TOPAS Symbolic system; the overriding plus is the flexibility allowed. Dependencies are automatically taken care of and unnecessary recalculations kept to a minimum. QUANT-1.INP uses many of the above keywords and additionally writes equivalent terms in the form of equations, for example:

```
prm = 100 Get(smv) / Get(sum_smvs); : 0 ' This is weight_percent

prm q = spiked_phase_measured_weight_percent /
        spiked_phase_measured_weight_percent_wt; : 0

prm = q Get(weight_percent); : 0    ' This is corrected_weight_percent
prm = 100 (1 - q); : 0              ' This is weight_percent_amorphous
```

14.1 ... Elemental weight percent constraint

If an elemental weight percent was known, and three phases of the mixture comprised this element then Get_Element_Weight can be used to get the weight of the element as a function of the structure; i.e.

```
    str ...
        prm z1 = Get_Element_Weight(Zr);
        MVW(!m1 0, !v1 0,0)
    str ...
        scale s2 0.001
        prm z2 = Get_Element_Weight(Zr);
        MVW(0, !v2 0,0)
    str ...
        scale s3 0.001
        prm z3 = Get_Element_Weight(Zr);
        MVW(0, !v3 0,0)
```

Rearranging the formulae for element weight percent, the scale parameter of one of the phases, say the first one, can be written as:

```
    scale = (0.01 known_Zr Get(sum_smvs_minus_this) - s2 v2 z2 - s3 v3 z3)
            / (v1 (z1 - 0.01 known_Zr m1));
```

Get(sum_smvs_minus_this) returns the sum of SMVs minus the phase where it is defined. QUANT-3.INP demonstrates this constraint with good convergence. It comprises 4 phases with three comprising Zr atoms. QUANT-2.INP demonstrates constraining a weight percent to a known value using the macro:

```
macro Known_Weight_Percent(& w)
{
   scale = (w / (100 - w)) Get(sum_smvs_minus_this) / (Get(cell_mass) Get(cell_volume));
}
```

14.2 ... Elemental composition and Restraints

The *xdd* dependent *elemental_composition* reports on the elemental composition for atoms within the structures of the *xdd*, for example:

```
    ' Before refinement
    xdd ...
        elemental_composition
```

```
    ' After refinement
    xdd ...
        elemental_composition
            {
                               Rietveld
                AL            0.875`_0.021
                O            26.135`_0.009
                SI            0.090`_0.003
                Y             6.289`_0.012
                ZR           66.612`_0.029
            }
```

The *xdd* dependent *elemental_weight_percent* returns the weight percent of an element within the corresponding *str*'s of the *xdd*. Example usage:

```
' Before refinement                    ' After refinement
penalties_weighting_K1 0.1             penalties_weighting_K1 0.1
xdd ...                                xdd ...
    element_weight_percent Zr+4 zr  0      element_weight_percent Zr+4 zr 65.027
    restraint = (zr - 65); : 0             restraint = (zr - 65); : 0.027525
```

In this example zr is the name given to the element Zr+4, the restraint shows a known value of 65 (set for example by XRF results). The refinement obeys the restraint according to the value set for *penalties_weighting_K1*. A weight percent can be restrained using:

```
xdd ...
    penalties_weighting_K1 0.2
    restraint = (Cubic_Zirconia_wt_percent - 36); : 0
    str ...
        MVW(0,0, !Cubic_Zirconia_wt_percent 0)
```

Note the name 'Cubic_Zirconia_wt_percent' which is given to *weight_percent*.

14.3 ... Amorphous phase composition

If *spiked_phase_measured_weight_percent* is defined then *elemental_composition* will report on Rietveld values, Corrected values and values from the original un-spiked sample. If *element_weight_percent_known* keywords are defined then *elemental_composition* will additionally report on the elemental contents of the amorphous phase, for example, from QUANT-1.INP we have:

```
elemental_composition
{
                Rietveld          Corrected          Original            Other
    AL          1.176`_0.042      1.059`_0.000       0.000`_0.000        0.000`_0.000
    O          26.271`_0.017     23.640`_0.832      23.162`_0.849        0.838`_0.849
    SI          0.104`_0.004      0.094`_0.005       0.096`_0.005        0.000`_0.000
    Y           6.182`_0.013      5.563`_0.204       5.676`_0.209        0.000`_0.000
    ZR         66.267`_0.055     59.631`_2.185      60.847`_2.229        2.153`_2.229
    Other       0.000`_0.000     10.015`_3.224      10.219`_3.290        7.228`_0.212
}
```

The 'Rietveld' and 'Corrected' columns corresponds to elemental weight-percents as determined for the spiked phase; the 'Original' and 'Other' columns corresponds to elemental weight-percents of the original phase. The 'Rietveld', 'Corrected' and 'Original' columns sum to 100%. The last row of the 'Corrected' column (purple number) corresponds to Get(*weight_percent_amorphous*). The last row of the 'Other' column (red number) is the amount of sample that is undefined; it comprises the Green number minus the elements of the 'Other' column. Note the zeros for Al (blue number); this is due to the spiked phase (dummy test data) being the only phase containing Al.

14.4 ... Using a dummy_str phase to describe amorphous content

If it is known that the amorphous content (purple number) in the above table comprises a known composition, say TiO_2, then a *dummy_str* can be used to describe the amorphous content, or:

```
dummy_str
```

```
      phase_name "Amorphous"
      a 5 b 5 c 5
      space_group 1
      site Ti occ Ti 1
      site O occ O 2
      Known_Weight_Percent(10.0148)
      MVW(0, 0 ,0)
```

dummy_str's that are void of *MVW* takes no part in Quantitative analysis. However, its lattice parameters and chemistry should correspond to a real structure so that *Mixture_LAC_1_on_cm* and *phase_LAC* can be correctly calculated; in the case of using the Brindley correction these changed values will change the quantitative results. The space group entry can be different to *P1* so long as the chemistry is correct. Inclusion of the *dummy_str* produces:

```
      elemental_composition
      {
                        Rietveld          Corrected          Original
      AL            1.059`_0.038      1.059`_0.000       0.000`_0.000
      O            27.652`_0.015     27.652`_0.975      27.256`_0.995
      SI            0.094`_0.003      0.094`_0.005       0.096`_0.005
      TI            6.002`_0.000      6.002`_0.215       6.125`_0.219
      Y             5.563`_0.012      5.563`_0.204       5.676`_0.209
      ZR           59.631`_0.050     59.631`_2.185      60.847`_2.229
      Other         0.000`_0.000      0.000`_3.583       0.000`_3.656
      }
```

Note that the 'Other' row becomes zero as the amorphous content is assigned to the *dummy_str*. The change in mixture values are:

Without *dummy_str*:

```
      Mixture_LAC_1_on_cm( 557.47740`_0.58665)
      mixture_density_g_on_cm3  5.26713308`_0.00292681843
```

With *dummy_str*:

```
      Mixture_LAC_1_on_cm( 608.85143`_0.76954)
      mixture_density_g_on_cm3  5.86601008`_0.00407998952
```

If XRF results were entered for *element_weight_percent_known*, for example:

```
      element_weight_percent_known Zr 63
      element_weight_percent_known O  24
```

then we get:

```
    elemental_composition
    {
                    Rietveld           Corrected            Original               Other
      AL          1.059`_0.038       1.059`_0.000       0.000`_0.000         0.000`_0.000
      O          27.652`_0.015      27.652`_0.975      27.256`_0.995        -3.256`_0.995
      SI          0.094`_0.003       0.094`_0.005       0.096`_0.005         0.000`_0.000
      TI          6.002`_0.000       6.002`_0.215       6.125`_0.219         0.000`_0.000
      Y           5.563`_0.012       5.563`_0.204       5.676`_0.209         0.000`_0.000
      ZR         59.631`_0.050      59.631`_2.185      60.847`_2.229         2.153`_2.229
      Other       0.000`_0.000       0.000`_3.583       0.000`_3.656         1.103`_0.431
    }
```

The negative element weight percent for O for the amorphous content reflects the fact that the measured XRF value for O is lower than the refinement's value (this example is used for testing; XRF values are fictitious).

14.5 … Quant using hkl_Is or other non-str phases

dummy_str's can be used to represent the quantitative results arising from non-str phases. For example, consider a phase where the structure is not known but the chemistry is known. If a calibration constant has been determined relating the *hkl_Is* intensities to the *scale* parameter of the *hkl_Is* phase, then a *dummy_str* can be written as follows (see QUANT-6.INP):

```
    dummy_str
        phase_name "Linked Cubic Zirconia"
        Cubic(5.137866)
        space_group F_M_-3_M
        site Zr x 0     y 0     z 0       occ  Zr  0.85
                                          occ  Y   0.15
        site O  x 0.25 y 0.25 z 0.25   occ  O   0.96
        scale = hkl_scale;
        Phase_LAC_1_on_cm(0)
        Phase_Density_g_on_cm3(0)
        MVW(0, 0 ,0)
```

Note, in this case a space group has been entered with structural parameters that looks like a known structure; this could, for example, occur where the structure is known in an ordered state, but the diffraction pattern comprises a disordered state. In other cases, the *P1* space group may suffice with site occupancies corresponding to the appropriate chemistry. The *dummy_str* is linked to the *hkl_Is* phase by assigning the *scale* parameter of the *dummy_str* to the *scale* parameters of the *hkl_Is* phase. QUANT-7.INP is a similar except that a *fit_obj* is linked to a *dummy_str*. Graphically the linked *dummy_str* is plotted with the calculated pattern of the *hkl_Is* phase or *fit_obj*, for example, QUANT-7.INP produces:
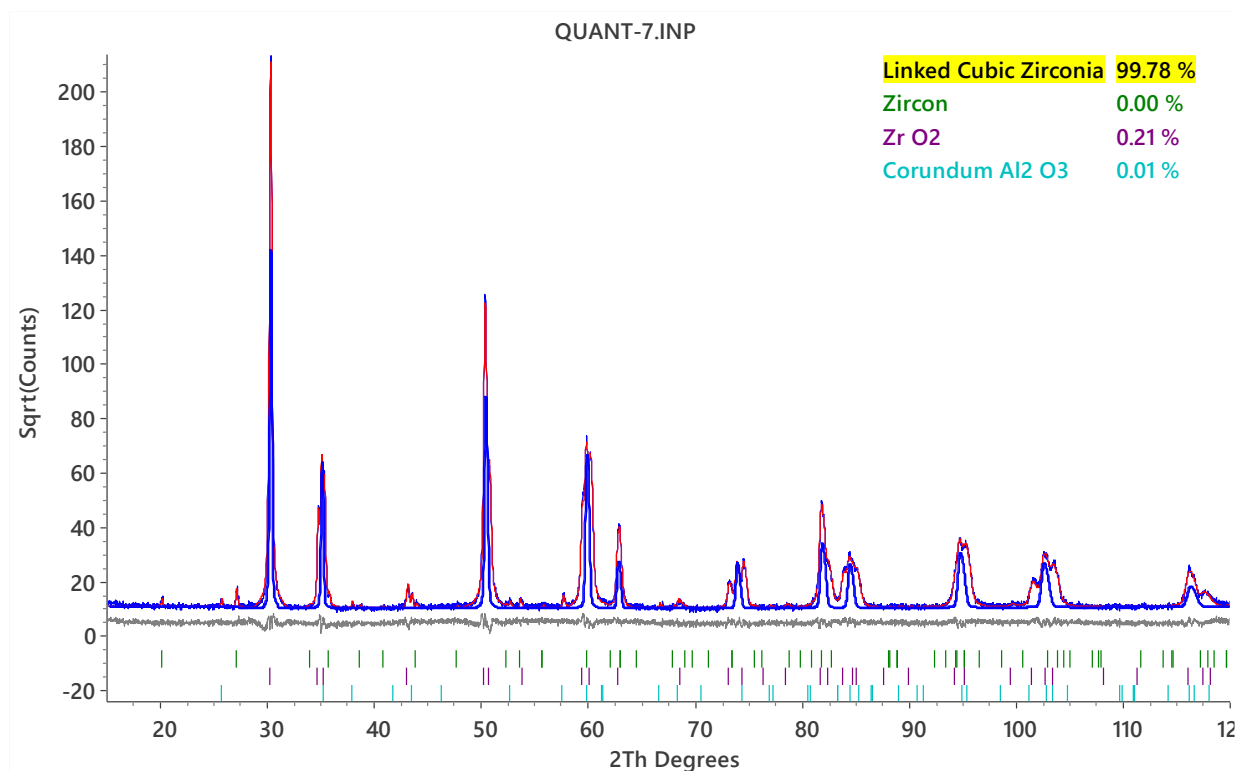
Here the blue line corresponds to the *dummy_str* which plots the calculated pattern of the linked *fit_obj* which in turn comprises a *user_y* object. The weight percent value determined by the *dummy_str* is also displayed.

14.6 ... Summary of Quant examples

- QUANT-1.INP: shows the use of *element_weight_percent_known* etc.
- QUANT-2.INP: uses the *Known_Weight_Percent* macro
- QUANT-3.INP: uses elemental constrain using Get_Element_Weight
- QUANT-4.INP: uses *Known_Weight_Percent* on a *hkl_ls* phase.
- QUANT-5.INP: uses a *dummy_str* to describe an amorphous phase
- QUANT-6.INP: uses a *hkl_ls* phase; links a *dummy_str* to the *hkl_ls* phase.
- QUANT-7.INP: uses a *fit_obj* that is a function of a *user_y* object to describe a phase; links a *dummy_str* to a *fit_obj* to get QUANT info.

14.7 ... External standard method

The method of O'Connor and Raven (1988) has been implemented in both GUI and Launch modes using the macros (see TEST_EXAMPLES\K-FACTOR):

```
macro K_Factor_MAC_K(mac, k, tot) {
    move_to xdd
    local !k_factor_mac_local_ mac
    local !k_factor_k_local_ k
    local !k_factor_sum_wps_ = 0; : tot
}
```

```
    macro K_Factor_WP(result) {
       local k_factor_wp_ = 1.6605402 Get(smv) k_factor_mac_local_
                          / k_factor_k_local_; : result
       if Prm_There(k_factor_sum_wps_) {
          existing_prm k_factor_sum_wps_ += k_factor_wp_;
       }
    }
```

14.8 ... QUANT Keywords

[*cell_mass* !E][*cell_volume* !E][*weight_percent* !E]
   [*spiked_phase_measured_weight_percent* !E][*corrected_weight_percent* !E]

cell_mass, *cell_volume* and *weight_percent* correspond to unit cell mass, volume and weight percent of the phase within the mixture. *spiked_phase_measured_weight_percent* defines the weight percent of a spiked phase. It is used by the *xdd* dependent *weight_percent_amorphous* to determine amorphous weight percent. Only one phase per *xdd* is allowed to contain *spiked_phase_measured_weight_percent*. *corrected_weight_percent* is the weight percent after considering amorphous content as determined by *weight_percent_amorphous*. The weight fraction w_p for phase p is calculated as follows:

$$w_p = \frac{Q_p}{\sum_{p=1}^{N_p} Q_p}$$

where N_p = Number of phases.

$Q_p = S_p M_p V_p / B_p$

S_p = Rietveld scale factor for phase p.

M_p = Unit cell mass for phase p.

V_p = Unit cell volume for phase p.

B_p = Brindley correction for phase p,

The Brindley correction is a function of *brindley_spherical_r_cm* and the phase and mixture linear absorption coefficients; the latter two are in turn functions of *phase_MAC* and *mixture_MAC* respectively, or,

B_p is function of : $(\text{LAC}_{\text{phase}} - \text{MAC}_{\text{mixture}})$ *brindley_spherical_r_cm*

$\text{LAC}_{\text{phase}}$ = linear absorption coefficient of phase p, packing density=1.

$\text{MAC}_{\text{mixture}}$ = linear absorption coefficient of the mixture, packing density=1.

This makes B_p a function of the weight fractions w_p of all phases and thus w_p as written above cannot be solved analytically. Subsequently w_p is solved numerically through the use of iteration.

[*mixture_density_g_on_cm3* #]

Calculates the density of the mixture assuming a packing density of 1, see also *mixture_MAC*.

[*mixture_MAC* #]

Calculates the mass absorption coefficient in cm^2/g for a mixture as follows:

$$\left(\frac{\mu}{\rho}\right)_{mixture} = \sum_{i=1}^{N} \left(\frac{\mu}{\rho}\right)_{i} w_i$$

where w_i and $(\mu/\rho)_i$ is the weight percent and *phase_MAC* of phase *i* respectively. Errors are reported for *phase_MAC* and *mixture_MAC*. The following example provides phase and mixture mass absorption coefficients.

```
xdd ...
    mixture_MAC 0
    str ...
        phase_MAC 0
```

The macros *Mixture_LAC_1_on_cm*, *Phase_LAC_1_on_cm* and *Phase_Density_g_on_cm3* calculates the mixture and phase linear absorption coefficients (for a packing density of 1) and phase density, for example:

```
xdd ...
    Mixture_LAC_1_on_cm(0)
    str ...
        Phase_Density_g_on_cm3(0)
        Phase_LAC_1_on_cm(0)
```

Errors for these quantities are also calculated. Mass absorption coefficients obtained from NIST at http://physics.nist.gov/PhysRefData/XrayMassCoef are used to calculate *mixture_MAC* and *phase_MAC*.

[*phase_MAC* !E]

Calculates the mass absorption coefficient in cm^2/g for the current phase. See description for *mixture_MAC*.

[*weight_percent_amorphous* !E]

Determines the amorphous content in a sample. The phase dependent *spiked_phase_measured_weight_percent* needs to be defined for *weight_percent_amorphous* to be calculated.

15. MAGNETIC STRUCTURE REFINEMENT

[*str*]...
    [*mag_only_for_mag_sites*]
    [*mag_space_group* $symbol]
    [*site*]...
        [*mlx* E][*mly* E][*mlz* E][*mg* E]
        [*mag_only*]
        'Site dependent macros
        MM_CrystalAxis_Display(mxc, myc, mzc)
        MM_CrystalAxis_Refine(mxc, mxv, myc, myv, mzc, mzv, mlx_v, mly_v, mlz_v)
        MM_Cartesian_Display(mxc, myc, mzc)
        MM_Cartesian_Refine(mxc, mxv, myc, myv, mzc, mzv, mlx_v, mly_v, mlz_v)

Thanks to Branton Campbell and John Evans for expert assistance during the implementation of magnetic refinement. Magnetic refinement is implemented using the keywords *mlx*, *mly*, *mlz*, *mg* and *mag_space_group*. See examples in the TEST_EXAMPLES\MAG directory as well as the tutorial by John Evans at:

    http://www.dur.ac.uk/john.evans/topas_workshop/tutorial_lamno3_magnetic.htm

The Magnetic intensity is given by (***** denotes conjugate gradient):

    Magnetic intensity = **Fmagcperp . Fmagcperp*** = |**Fmagcperp**|

    Fmagcperp = **Fmagc** – (**Fmagc** . **Qhat**) **Qhat**

Or in words, **Fmagcperp** is the component of the magnetic vector in the direction perpendicular to the scattering vector **Q**, where:

    $\mathbf{Q} = (\mathbf{L}^{-1})^{\mathrm{T}} * \mathbf{h}$

    Qhat = Q / |Q|

    L is the Cartesian lattice parameters in 3x3 matrix form

    h is the Miller indices in vector form

    * denotes matrix multiplication

    Superscript $^{-1}$ denotes matrix inverse

    Superscript $^{\mathrm{T}}$ denotes matrix transpose

    $(\mathbf{L}^{-1})^{\mathrm{T}}$ = reciprocal lattice parameters

Fmagc in terms of the Cartesian lattice parameters is:

    Fmagc = **L** * **Fmag**

Fmag for the plane **h** for a single site is:

    $\mathbf{Fmag} = \sum_j (\mathbf{B}_j * \mathbf{m}) \, \mathrm{Exp}(2\pi \, i \, U_j)$

where the summation is over the equivalent positions j and:

$U_j = \mathbf{h.R}_j \, \mathbf{x} + \mathbf{h.t}_j$

$\mathbf{x} = \{\, x, y, z \,\} = $ site fractional coordinates

$\mathbf{m} = \{\, mlx, mly, mlz \,\} = $ magnetic moment

$\mathbf{R}_j = $ rotation part of space group operator

$\mathbf{t}_j = $ translational part of space group operator

$d_j = s_j \, \text{determinant}(\mathbf{R}_j) = s_j \, \text{det}(\mathbf{R}_j)$

$\mathbf{B}_j = s_j \, \text{det}(\mathbf{R}_j) \, \mathbf{R}_j = $ magnetic transformation matrix

The file MAGDATA.DAT (a GSAS file - permission for use granted by Robert Von Dreele, author of GSAS) comprises data for calculating magnetic form factors. The Lande splitting factor can be refined using the site dependent parameter *mg*; defaults for *mg* are obtained from MAGDATA.DAT. Shubnikov groups are obtained from the file SHUBNIKOVGROUPS.TXT. When *mag_only* is defined, the x-ray component to intensity for the site in question is ignored. When *mag_only_for_mag_sites* is defined the x-ray component to intensity for all magnetic sites for the *str* in question is ignored.

15.1 ... Magnetic refinement warnings/exceptions

The following two messages:

1) Warning: Magnetic moment mlx of site Fe has no contribution to Fmag

2) Magnetic moment mlx of site Fe cannot be refined as it has no derivative

arise when for each group of equivalent positions of a special position the first row of the matrix $\sum_j \mathbf{B}_j {}^*\mathbf{m}$ is zero where the *j*'s sum over the equivalent positions of a special position group. Similar messages for *mly* and *mlz* are given. Note, the fact that *mlx*, *mly*, *mlz* may or may not be refined and their associated constraints are considered. Refinement terminates in the case of message (2) when *mlx* is being refined.

15.2 ... Displaying Magnetic moments

Magnetic moments (Occupancy $\mathbf{B}_j {}^*\mathbf{m}$) are displayed graphically when *view_structure* is defined. For the case where the atom balls are masking the display of the magnetic moment arrows, the "Atom size" can be varied as shown in the following:
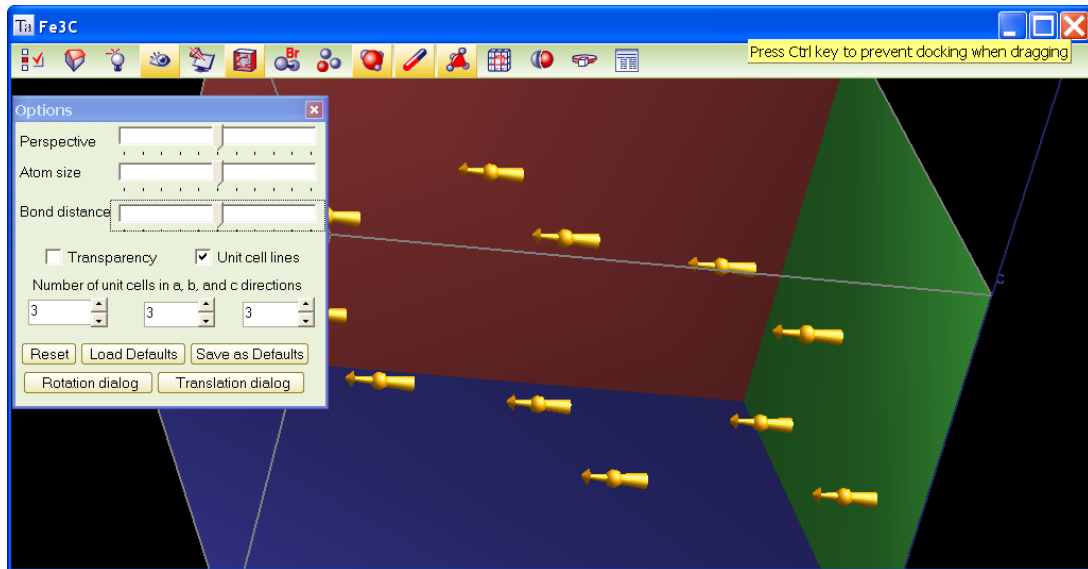
15.3 … 'Decomposing' Fmag for speed

When using magnetic space groups, equivalent positions for space groups other than 1.1 are written in terms of other equivalent positions.

Let $C_j = \cos(U_j)$,

$\quad S_j = \sin(U_j)$

$\quad \text{Exp}(iU) = C_j + iS_j = \text{Euler's formulae}$

For two equivalent positions of a special position we have:

$\quad U_1 = U_2 = U$

$\qquad \textbf{Fmag}_1 + \textbf{Fmag}_2 \quad = s_1 \det(\textbf{R}_1)\ \textbf{R}_1\ \textbf{m}\ \text{Exp}(iU) + s_2 \det(\textbf{R}_2)\ \textbf{R}_2\ \textbf{m}\ \text{Exp}(iU)$

$\qquad\qquad\qquad\qquad\quad = (s_1 \det(\textbf{R}_1)\ \textbf{R}_1 + s_2 \det(\textbf{R}_2)\ \textbf{R}_2)\ \textbf{m}\ \text{Exp}(iU)$

$\qquad\qquad\qquad\qquad\quad = \textbf{c}\ \textbf{m}\ \text{Exp}(iU)$

c is independent of **x**. Note, a particular special-position could have many equivalent positions.

If $\textbf{R}_1 = -\textbf{R}_2$ and $\textbf{t}_1 = -\textbf{t}_2$ for two equivalent positions then:

$\quad U_1 = -U_2 = U$

$\textbf{Fmag}_1 + \textbf{Fmag}_2 =$	$s_1 \det(\textbf{R})\ \textbf{R}\ \textbf{m}\ \text{Exp}(iU) + s_2 \det(-\textbf{R})(-\textbf{R})\ \textbf{m}\ \text{Exp}(-iU)$
Now,	$\det(\textbf{R})\ \textbf{R} = \det(-\textbf{R})(-\textbf{R})$
or,	$\textbf{Fmag}_1 + \textbf{Fmag}_2 = \det(\textbf{R})\ \textbf{R}\ \textbf{m}\ (s_1\,\text{Exp}(iU) + s_2\,\text{Exp}(-iU))$
For $s_1 = s_2$,	$\textbf{Fmag}_1 + \textbf{Fmag}_2 = s_1 \det(\textbf{R})\ \textbf{R}\ \textbf{m}\ 2\,C$
For $s_1 = -s_2$,	$\textbf{Fmag}_1 + \textbf{Fmag}_2 = s_1 \det(\textbf{R})\ \textbf{R}\ \textbf{m}\ (2\,i\,S)$

If $\mathbf{R}_1 = \mathbf{R}_2$ for two equivalent positions, then:

$$\mathbf{Fmag}_1 + \mathbf{Fmag}_2 \quad = s_1 \det(\mathbf{R})\,\mathbf{R}\,\mathbf{m}\,\text{Exp}(i\,\mathbf{h}.\,\mathbf{R}\,\mathbf{x})\,\text{Exp}(i\,\mathbf{h}.t_1) +$$

$$s_2 \det(\mathbf{R})\,\mathbf{R}\,\mathbf{m}\,\text{Exp}(i\,\mathbf{h}.\,\mathbf{R}\,\mathbf{x})\,\text{Exp}(i\,\mathbf{h}.t_2)$$

$$= \det(\mathbf{R})\,\mathbf{R}\,\mathbf{m}\,(s_1\,\text{Exp}(i\,\mathbf{h}.t_1) + s_2\,\text{Exp}(i\,\mathbf{h}.t_2))\,\text{Exp}(i\,\mathbf{h}.\,\mathbf{R}\,\mathbf{x})$$

$$= \mathbf{c}\,\text{Exp}(i\,\mathbf{h}.\,\mathbf{R}\,\mathbf{x})$$

$\mathbf{c}$ is independent of $\mathbf{x}$ and is calculated only once. Many $\mathbf{R}$'s can be the same for a particular space group with only the $\mathbf{t}$'s changing.

Calculating C and S:

$$\text{Exp}(i\,(\mathbf{h}.\,\mathbf{R}\,\mathbf{x} + \mathbf{h}.\,\mathbf{t})) = \text{Exp}(i\,\mathbf{h}.\,\mathbf{R}\,\mathbf{x})\,\text{Exp}(\,i\,\mathbf{h}.\,\mathbf{t})$$

$\text{Exp}(i\,\mathbf{h}.\,\mathbf{t})$ is constant for a particular $\mathbf{h}$ and is calculated only once.

Only unique $\text{Exp}(i\,\mathbf{h}.\,\mathbf{R}\,\mathbf{x})$ are calculated.

Trigonometric recurrence is used to calculate sines and cosines resulting in three cosine and three sine operations per unique equivalent r. In other words, a sin and cos are not calculated for each h. Note a sin or cos function is equivalent to about 40 to 60 multiplies.

16. RIGID BODIES

[*rigid*]...
    [*point_for_site* $site [*ux* | *ua* E] [*uy* | *ub E*] [*uz* | *uc* E]]...
        [*in_cartesian*] [*in_FC*]
    [*z_matrix* atom_1 [atom_2 E] [atom_3 E] [atom_4 E]]...
    [*rotate* E [*qx* | *qa* E] [*qy* | *qb* E] [*qz* | *qc* E]]...
        [*operate_on_points* $sites]
        [*in_cartesian*] [*in_FC*]
    [*translate* [*tx* | *ta* E] [*ty* | *tb* E] [*tz* | *tc* E]]...
        [*operate_on_points* $sites]
        [*in_cartesian*] [*in_FC*]
        [*rand_xyz* !E]
        [*start_values_from_site* $unique_site_name]

Rietveld or Pair Distribution Function refinement can comprise rigid bodies. Rigid bodies comprise points in space defined using *z_matrix* or *point_for_site* keywords or both simultaneously. All or some of these points can be operated on using *rotate* and *translate*. Rigid body operations include:

- Translating a rigid body or part of a rigid body.
- Rotating a rigid body or part of a rigid body around a point.
- Rotating a rigid body or part of a rigid body around a line.

ua, *ub*, *uc*, *ta*, *tb*, *tc*, *qa*, *qb*, *qc* and the parameters of *z_matrix* are all refineable parameters which can comprise parameter attributes such as *min*/*max*. The directory RIGID contains rigid body examples in *.RGD files. These files can be viewed and modified using the Rigid-Body-Editor of the GUI.

rigid defines the start of a rigid body. *point_for_site* defines a point in space with Cartesian coordinates given by the parameters *ux*, *uy uz*. Fractional equivalents can be defined using *ua*, *ub* and *uc*. $site is the *site* that the *point_for_site* represents. *z_matrix* defines a point in space with coordinates given in Z-matrix format as follows:

- E can be an equation, constant or a parameter name with a value.
- atom_1 specifies the site that the new Z-matrix point represents.
- The E after atom_2 specifies the distance in Å between atom_2 and atom_1. atom_2 must exist if atom_1 is preceded by at least one point.
- The E after atom_3 specifies the angle in degrees between atom_3, atom_2 and atom_1. atom_3 must exist if atom_1 is preceded by at least two points.
- The E atom_4 specifies the dihedral angle in degrees between the plane formed by atom_3-atom_2-atom_1 and the plane formed by atom_4-atom_3-atom_2. This angle is drawn using the righthand rule with the thumb pointing in the direction atom_3 to atom_2. atom_4 must exist if atom_1 is preceded by at least three sites of the rigid body.
- If atom_1 is the first point of the rigid body then it is placed at Cartesian (0, 0, 0). If atom_1 is the second point of the rigid body then it is placed on the positive z-axis at Cartesian (0, 0, E)

where E corresponds to the E in [atom_2 E]. If $atom_1 is the third point of the rigid-body then it is placed in the x-y plane.

rotate rotates *point_for_site*'s an amount as defined by the *rotate* E equation around the vector defined by the Cartesian vector *qx*, *qy*, *qz*. The vector can instead be defined in fractional coordinates using *qa*, *qb* and *qc*. *translate* performs a translation of *point_for_site*'s an amount in Cartesian coordinates equal to *tx*, *ty*, *tz*. The amount can instead be defined in fractional coordinates using *ta*, *tb* and *tc*. *rotate* and *translate* operates on any previously defined *point_for_site*'s; alternatively, *point_for_site*'s operated on can be identified using *operate_on_points*. *operate_on_points* must refer to previously defined *point_for_site*'s (see section 17.20 for a description of how to identify sites). *in_cartesian* or *in_FC* can be used to signal coordinates are in Cartesian or fractional atomic coordinates, respectively. When *continue_after_convergence* is defined, *rand_xyz* processes are initiated after convergence. It introduces a random displacement to the translate fractional coordinates (*tx*, *ty*, *tz*) that are independent parameters. The size of the random displacement is given by the current *temperature* multiplied by #displacement where #displacement is in Å. *start_values_from_site* initializes the values *ta*, *tb*, *tc* with corresponding values taken from the site $unique_site_name.

16.1 ... Fractional, Cartesian and Z-matrix coordinates

Rigid bodies can be formulated using fractional or Cartesian coordinates. A Benzene ring without Hydrogens can be formulated as follows:

```
prm a 1.3 min 1.2 max 1.4
rigid
   point_for_site C1 ux =  a Sqrt(3) .5; uy =  a .5;
   point_for_site C2 ux =  a Sqrt(3) .5; uy = -a .5;
   point_for_site C3 ux = -a Sqrt(3) .5; uy =  a .5;
   point_for_site C4 ux = -a Sqrt(3) .5; uy = -a .5;
   point_for_site C5 uy =  a;
   point_for_site C6 uy = -a;
   Rotate_about_axies(@ 0, @ 0, @ 0) ' rotate previously defined points
   Translate(@ 0.1, @ 0.2, @ 0.3)    ' translate previously defined points
```

The last two statements rotate and translates the rigid body as a whole; their inclusion is implied if absent. A formulation of any complexity can be obtained from a) databases of existing structures using fractional or Cartesian coordinates of structure fragments or b) from sketch programs for drawing chemical structures. A Z-matrix representation of a rigid body explicitly defines the rigid body in terms of bond lengths and angles. A Benzene ring is typically formulated using two dummy atoms *X1* and *X2* as follows:

```
    str ...
        site X1 ... occ C 0
        site X2 ... occ C 0
        rigid
            load z_matrix {
                X1
                X2    X1   1.0
                C1    X2   1.3    X1   90
                C2    X2   1.3    X1   90   C1   60
                C3    X2   1.3    X1   90   C2   60
                C4    X2   1.3    X1   90   C3   60
                C5    X2   1.3    X1   90   C4   60
                C6    X2   1.3    X1   90   C5   60
            }
```

Atoms with occupancies fixed to zero are dummy atoms and do not take part in structure factor calculations. Importantly however dummy atoms take part in penalties. The mixing of *point_for_site* and *z_matrix* keywords is possible as follows:

```
    rigid
        point_for_site X1
        load z_matrix {
            X2    X1   1.0
            C1    X2   1.3    X1   90 ...
        }
```

Z-matrix parameters are like any other parameter; they can be equations and parameter attributes can be assigned. For example, the 1.3 bond distance can be refined as follows:

```
    rigid
        point_for_site X1
        load z_matrix {
            X2    X1   1.0
            C1    X2   c1c2 1.3 min 1.2 max 1.4  X1   90
            C2    X2   = c1c2;   X1   90   C1   60
            C3    X2   = c1c2;   X1   90   C2   60
            C4    X2   = c1c2;   X1   90   C3   60
            C5    X2   = c1c2;   X1   90   C4   60
            C6    X2   = c1c2;   X1   90   C5   60
        }
```

This ability to constrain Z-matrix parameters using equations allow for great flexibility. Example use could involve writing a Z-matrix bond length parameter in terms of other bond length parameters whereby the average bond length is maintained. Or, in cases where a bond length is expected to change as a function of a site occupancy, an equation relating the bond length as a function of the site occupancy parameter can be formulated.

16.2 ... Translating part of a rigid body

Once a starting rigid body model is defined, further *translate* and *rotate* statements can be included to represent deviations from the starting model. For example, if the C1 and C2 atoms are expected to shift by up to 0.1Å and as a unit then the following could be used:

```
rigid
   load z_matrix {
      X1
      X2   X1  1.0
      C1   X2  1.3   X1  90
      C2   X2  1.3   X1  90  C1  60
      C3   X2  1.3   X1  90  C2  60
      C4   X2  1.3   X1  90  C3  60
      C5   X2  1.3   X1  90  C4  60
      C6   X2  1.3   X1  90  C5  60
   }
   translate
      tx @ 0 min -0.1 max 0.1
      ty @ 0 min -0.1 max 0.1
      tz @ 0 min -0.1 max 0.1
      operate_on_points "C1 C2"
```

where the additional statements are in purple. The Cartesian coordinate representation allows an additional means of shifting the C1 and C2 atoms by refining on the *ux*, *uy* and *uz* coordinates directly, or,

```
prm a 1.3 min 1.2 max 1.4
prm t1 0 min -0.1 max 0.1
prm t2 0 min -0.1 max 0.1
prm t3 0 min -0.1 max 0.1
rigid
   point_for_site C1 ux = a Sqrt(3) 0.5 + t1; uy =  a 0.5 + t2; uz = t3;
   point_for_site C2 ux = a Sqrt(3) 0.5 + t1; uy = -a 0.5 + t2; uz = t3;
   point_for_site C3 ux =-a Sqrt(3) 0.5;      uy =  a 0.5;
   point_for_site C4 ux =-a Sqrt(3) 0.5;      uy = -a 0.5;
   point_for_site C5                          uy =  a;
   point_for_site C6                          uy = -a;
```

16.3 ... Rotating part of a rigid body around a point

Many situations require the rotation of part of a rigid body around a point. An octahedra (Fig. 16-1) for example typically rotates around the central atom with three degrees of freedom. To implement such a rotation when the central atom is arbitrarily placed requires setting the origin at the central atom before rotation and then resetting the origin after rotation. This is achieved using the *Translate_point_amount* macro as follows:

```
prm r 2 min 1.8 max 2.2
rigid
   point_for_site A0
   point_for_site A1 ux =  r;
   point_for_site A2 ux = -r;
   point_for_site A3 uy =  r;
   point_for_site A4 uy = -r;
   point_for_site A5 uz =  r;
   point_for_site A6 uz = -r;
   Translate_point_amount(A0, -) operate_on_points "A* !A0"
   rotate @ 0 qa 1 operate_on_points "A* !A0"
   rotate @ 0 qb 1 operate_on_points "A* !A0"
   rotate @ 0 qc 1 operate_on_points "A* !A0"
```

```
          Translate_point_amount(A0, +) operate_on_points "A* !A0"
```

The *point_for_site keywords* could just as well be *z_matrix* keywords with the appropriate Z-matrix parameters. The first *Translate_point_amount* statement translates the specified points (A1 to A6) an amount equivalent to the negative position of A0. This sets the origin for these points to A0. The second resets the origin back to A0. If the A0 atom happens to be at Cartesian (0, 0, 0) then there would be no need for the *Translate_point_amount* statements.
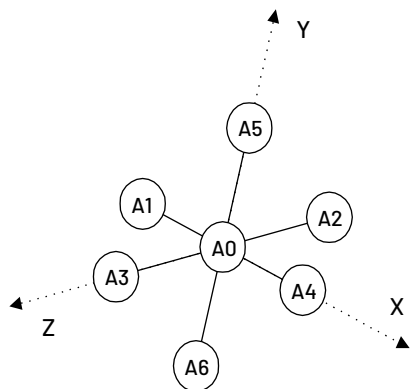
Fig. 16-1 Model of an ideal octahedron.

Further distortions are possible by refining on different bond-lengths between the central atom and selected outer atoms. For example, the following macro describes an orthorhombic bipyramid:

```
    macro Orthorhombic_Bipyramide(s0, s1, s2, s3, s4, s5, s6, r1, r2) {
       point_for_site s0
       point_for_site s1 ux    r1
       point_for_site s2 ux   −r1
       point_for_site s3 uy    r1
       point_for_site s4 uy   −r1
       point_for_site s5 uz    r2
       point_for_site s6 uz   −r2
    }
```

Note the two different lengths r1 and r2; with r1 = r2 this macro would describe a regular octahedron.

16.4 ... Rotating part of a rigid body around a line

Instead of explicitly entering fractional or Cartesian coordinates, rigid bodies can be created using the *rotate* and *translate* keywords. For example, two connected Benzene rings, a schematic without Hydrogens is shown in Fig. 16-2, can be formulated as follows:

```
    prm r 1.3 min 1.2 max 1.4
    rigid
       point_for_site C1 ux = r;
       load point_for_site ux rotate qz operate_on_points {
          C2 =r; 60  1 C2
          C3 =r; 120 1 C3
          C4 =r; 180 1 C4
          C5 =r; 240 1 C5
          C6 =r; 300 1 C6
```

```
        }
    point_for_site C7 ux = r;
    load point_for_site ux rotate qz operate_on_points {
        C8  =r;  60  1 C8
        C9  =r; 120  1 C9
        C10 =r; 300  1 C10
    }
    translate tx = 1.5 r; ty = r Sin(60 Deg);
        operate_on_points "C7 C8 C9 C10"
```

The points of the second ring can be rotated around the line connecting C1 to C2 with the following:

```
    Rotate_about_points(@ 50 min -60 max 60, C1, C2, "C7 C8 C9 C10")
```

The *min*/*max* statements limit the rotations to ±30 degrees. C5 can be rotated around the line connecting C4 and C6 with the following:

```
    Rotate_about_points(@ 40 min -50 max 50, C4, C6, C5)
```

Similar *Rotate_about_points* statements for each atom would allow for distortions of the Benzene rings without changing bond distances.
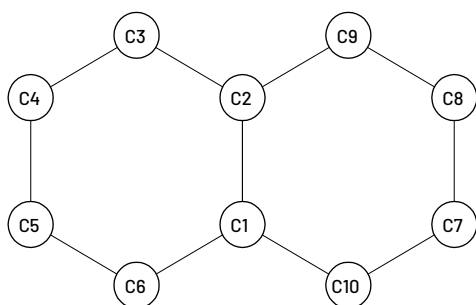
Fig. 16-2. Model of two connected Benzene rings

Another means of generating Fig. 16-2 and the one that requires the least thought is by using the *Duplicate_Point* and *Duplicate_rotate_z* macros as follows:

```
    prm r 1.3 min 1.2 max 1.4
    rigid
        point_for_site C1 ux = r;
        Duplicate_rotate_z(C2, C1, 60)
        Duplicate_rotate_z(C3, C2, 60)
        Duplicate_rotate_z(C4, C3, 60)
        Duplicate_rotate_z(C5, C4, 60)
        Duplicate_rotate_z(C6, C5, 60)
        Duplicate_Point(C7,  C3)
        Duplicate_Point(C8,  C4)
        Duplicate_Point(C9,  C5)
        Duplicate_Point(C10, C6)
        Rotate_about_points(180, C1, C2, "C7 C8 C9 C10")
```

16.4.1 ....Using Z-matrix together with rotate and translate

Cyclopentadienyl (C5H5) is a well-defined molecular fragment which shows slight deviations from a perfect five-fold ring (Fig. 16-3). The rigid body definition using *point_for_site* keywords is as follows:

```
prm r1 1.19
prm r2 2.24
rigid
   load point_for_site ux { C1 =r1; C2 =r1; C3 =r1; C4 =r1; C5 =r1; }
   load point_for_site ux { H1 =r2; H2 =r2; H3 =r2; H4 =r2; H5 =r2; }
   load rotate qz operate_on_points { 72 1 C2  144 1 C3 216 1 C4 288 1 C5 }
   load rotate qz operate_on_points { 72 1 H2  144 1 H3 216 1 H4 288 1 H5 }
```

and using a typical Z-matrix representation:

```
rigid
   load z_matrix {
      X1
      X2   X1 1
      C1   X2 1.19   X1  90
      C2   X2 1.19   X1  90   C1  72
      C3   X2 1.19   X1  90   C2  72
      C4   X2 1.19   X1  90   C3  72
      C5   X2 1.19   X1  90   C4  72
      X3   C1 1      X2  90   X1   0
      H1   C1 1.05   X3  90   X2 180
      H2   C2 1.05   C1 126   X2 180
      H3   C3 1.05   C2 126   X2 180
      H4   C4 1.05   C3 126   X2 180
      H5   C5 1.05   C4 126   X2 180
   }
```

This Z-matrix representation is typically used for Cyclopentadienyl; it allows for various torsion angles but it does not allow for all possibilities. For example, no adjustment of a single Z-matrix parameter allows for displacement of the C1 atom without changing the C1-C2 and C1-C3 bond distances. The desired result however is possible using the *Rotate_about_points* macro:

```
Rotate_about_points(@ 0, C2, C3, "C1 H1")
```

Thus, the ability to include *rotate* and *translate* together with *z_matrix* gives great flexibility in defining rigid bodies.
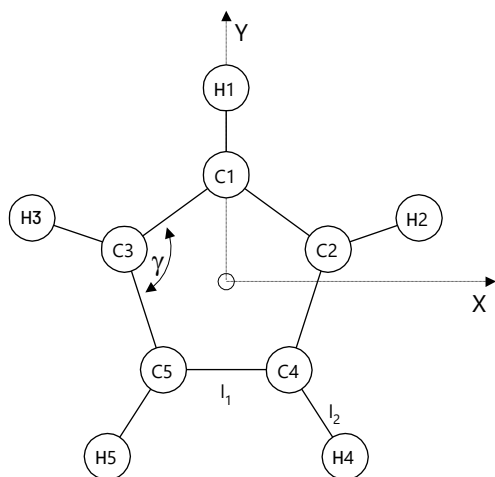
Fig. 16-3. Model of the idealized cyclopenta-dienyl anion (C_5H_5).

16.5 ... The simplest of rigid bodies

The simplest rigid body comprises an atom constrained to move within a sphere; for a radius of 1, this can be achieved as follows:

```
rigid
    point_for_site Ca uz @ 0 min -1 max 1
    rotate r1 10 qx 1
    rotate r2 10 qx = Sin(Deg r1); qy = -Cos(Deg r1);
```

The coordinates are in fact spherical coordinates where the rotation parameters r1 and r2 are communicative. Constraining an atom to within a sphere is a useful constraint for an atomic position when the approximate position is known. Setting the distance between two sites, or, two sites A and B a distance 2Å apart can be formulated as:

In Z-matrix form:
```
rigid
    z_matrix A                        ' line 1
    z_matrix B A 2                    ' line 2
    rotate @ 20 qa 1                  ' line 3
    rotate @ 20 qb 1                  ' line 4
    translate ta @ 0.1 tb @ 0.2 tc @ 0.3   ' line 5
```

In Cartesian form:
```
rigid
    point_for_site A                  ' line 1
    point_for_site B uz 2             ' line 2
    rotate @ 20 qa 1                  ' line 3
    rotate @ 20 qb 1                  ' line 4
    translate ta @ 0.1 tb @ 0.2 tc @ 0.3   ' line 5
```

Lines 1 and 2 defines the two points (note *ux*, *uy* and *uz* defaults to 0), line 3 and 4 rotates the two points around the a and then the b lattice vectors. Line 5 translates the two points to a position in fractional atomic coordinates of (0.1, 0.2, 0.3). Lines 3 to 5 contain the five parameters associated with this rigid body. The *Set_Length* macro can instead be used to set the distance between the two sites as follows:

```
Set_Length(A, B, 2, @, @, @, @ 30, @ 30)
```

where A and B are the site names, 2 is the distance in Å between the sites, arguments 4 to 6 the names given to the translation parameters and arguments 7 and 8 are the rotational parameters. *Set_Length* is not supplied with the *translate* starting values; these are obtained from the A site with the use of *start_values_from_site* located in the *Set_Length* macro. *min*/*max* can be used to constrain the distance between the two sites, for example:

```
Set_Length(A, B, @ 2 min 1.9 max 2.1, @, @, @, @ 30, @ 30)
```

Note, this macro defines the distance between the two sites as a parameter that can be refined.

16.6 ... Generation of rigid bodies

A rigid body is constructed by the sequential processing of *z_matrix*, *point_for_site*, *rotate* and *translate* operations. The body is then converted to fractional atomic coordinates and then symmetry operations of the space group applied. The conversion of Z-matrix coordinates to Cartesian is as follows:

- The first atom, if defined, is paced at the origin.
- The second atom, if defined, is placed on the positive z-axis.
- The third atom, if defined, is placed in the x-z plane.

For Cartesian to fractional coordinates in terms of the lattice vectors we have:

- x-axis in the same direction as the **a** lattice vector.
- y-axis in the **a**-**b** plane.
- z-axis in the direction defined by the cross product of **a** and **b**.

Rotation operations are not commutative; the rotation of point A about the vector B-C and then about D-E is not the same as the rotation of A about D-E and then about B-C. By default, *rotate* and *translate* operate on all previously defined *point_for_site*'s. Alternatively *point_for_site*'s can be explicitly defined using *operate_on_points. operate_on_points* must refer to previously defined *point_for_site*'s and it can refer to many sites at once by enclosing the site names in quotes and using the wild card character '*' or the negation character '!' (see section 17.20), for example:

```
operate_on_points "Si* O* !O2"
```

16.7 ... Rigid body parameter errors propagated to fractional coordinates

Errors for fractional coordinates for sites defined as part of a rigid body are propagated to the site fractional coordinates. The example RIGID-ERRORS\ANILINE_I_100K_X.INP (by Simon Parsons) demonstrates the equivalence of two refinements 1) using a rigid body and 2) hand coding the fractional coordinates in terms of rigid body parameters but not in fact using a rigid body. Errors and convergence behavior in both cases are identical. Case (2), which has many computer algebra equations, takes approximately the same time per iteration as case (1); this demonstrates that computer algebra does not noticeably affect computational speed even in cases where its use is plentiful.

16.8 ... Z-matrix collinear error information

The Z-matrix collinear points exception can be deciphered using information displayed on detection of the error. The collinear error is due to three atoms on a z-matrix line which are collinear. The information displayed includes a snapshot of the rigid body operations pertaining to the error. The following is an example of the information displayed:

```
DB_x_CB Zero dot product - Z-matrix possible collinear points at atoms
    O10
    C16 8.91631604e-016 1.0912987e-014 5.2
    C15 3.72315026e-016 1.0912987e-014 3.9
    C11 0 0 0
Partial z-matrix in error:
    rigid
        z_matrix C11
        z_matrix C12 C11 1.3
        z_matrix C13 C12 1.3 C11 120
        z_matrix C14 C13 1.3 C12 120 C11 180
        z_matrix C15 C14 1.3 C13 120 C11 0
        z_matrix C16 C15 1.3 C14 120 C11 180
        z_matrix O10 C16 1   C15 108 C11 120
```

The rigid body fragment can be copied to the Rigid-body editor to investigate why the error is occurring; i.e.
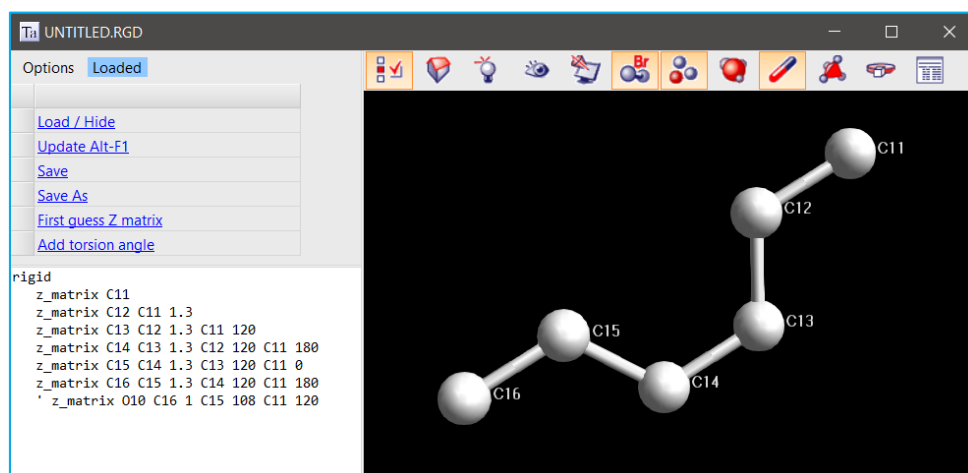
The O10 line is commented out as it's the line causing the error. Looking at the O10 line (using the OpenGL window), we see that atoms C16, C15, C11 lie on a straight line; this is invalid as it becomes impossible to form the dihedral angle in a non-degenerate manner. The best way to think about a z-matrix line with 4 atoms A, B, C, D, i.e.

```
        z_matrix A B # C # D #
```

is to think of two triangles ABC and DBC hinged along the line BC. The angle between the triangles is the dihedral angle. If B,C,D are collinear then there's no triangle and the dihedral angle cannot be formed. Thus, for z-matrices both A,B,C and B,C,D must not be collinear. The program tests for a zero dot-product numerically with a tolerance of 10^{-15}.

16.9 ... Functions allowing access to rigid-body fractional coordinates

The standard macro *Point*(site_name, rx), see TOPAS.INC, returns the x Cartesian coordinate of the point called site_name; y and z Cartesian coordinates are returned by ry and rz objects respectively. These functions can only to be used in equations of the rigid body which encompass the keywords and their dependents of *point_for_site*, *z_matrix*, *translate* and *rotate*. The actual value returned by *Point* depends on where it is used in the rigid-body, for example, in the following:

```
rigid
   point_for_site O1
   translate tx 1
   point_for_site O2 ux = Point(O1, rx); ' Point here returns 1
   translate tx 2
   point_for_site O3 ux = Point(O1, rx); ' Point here returns 3
```

the final x Cartesian coordinate of site O3 becomes 3. To instead return fractional coordinates of points, the functions Point_rx_ua, Point_rx_ub and and Point_rx_ua can be used. These functions are passed the address of the point in question using the *Point* macro with one argument. Accompanying macros simplifying the call, as defined in TOPAS.INC, are:

```
macro Point_ua(site_name) { Point_rx_to_ua(Point(site_name)) }
macro Point_ub(site_name) { Point_ry_to_ub(Point(site_name)) }
macro Point_uc(site_name) { Point_rz_to_uc(Point(site_name)) }
```

These macros can return many different values for the same point in question depending on where they are called during the rigid body calculation.

16.10 . Determining the orientation of a known fragment using a Rigid-Body

Determines rotation and translation parameters for a known fragment, see TEST_EXAMPLES\RIGID\MATCH.INP. The known fragment is in fractional coordinates. To do the same for a fragment in Cartesian coordinates then change the lattice angles to 90 degrees and adjust the lattice parameter lengths. Also, see:

http://topas.dur.ac.uk/topaswiki/doku.php?id=rigid_body_-_matching_to_a_known_fragment

16.11 . Rigid body macros

Set_Length(s0, s1, r, xc, yc, zc, cva, cvb)

    Fixes the distance between two sites.

    [s0, s1]: Site names.

    [r]: Distance in Å.

    [xc, yc, zc]: The parameter names for the coordinates of s0.

    [cva, cvb]: Parameter names and values for rotations about the x and y axes

Set_Lengths(s0, s1, s2, r, xc, yc, zc, cva1, cvb1, cva2, cvb2)
Set_Lengths(s0, s1, s2, s3, r, xcv, ycv, zcv, cva1, cvb1, cva2, cvb2, cva3, cvb3)

Sets the distance between two and three sites, respectively. The two sites case is defined as:

```
macro Set_Lengths(s0, s1, s2, r, xc, yc, zc, cva1, cvb1, cva2, cvb2)
{
    Set_Length(s0, s1, r, xc, yc, zc, cva1, cvb1)
    Set_Length(s0, s2, r, xc, yc, zc, cva2, cvb2)
}
```

Triangle(s1, s2, s3, r)
Triangle(s0, s1, s2, s3, r)
Triangle(s0, s1, s2, s3, r, xc, yc, zc, cva, cvb, cvc)

Defines a regular triangle without and with a central atom (s0).

[s0, s1, s2, s3]: Site names. s0 is the central atom of the triangle.

[r]: Distance in Å.

[xc, yc, zc]: Parameter names for the coordinates for the central atom.

[cva, cvb, cvc]: Parameter names and values for rotations about the x, y and z axes.

Tetrahedra(s0, s1, s2, s3, s4, r, xc, yc, zc, cva, cvb, cvc)

Defines a tetrahedra with a central atom.

[s0, s1, s2, s3, s4]: Site names. s0 is the central atom of the tetrahedra.

[r]: Distance in Å.

[xc, yc, zc]: Parameter names for the coordinates for the central atom.

[cva, cvb, cvc]: Parameter names and values for rotations about the x, y and z axes.

Octahedra(s0, s1, s2, s3, s4, s5, s6, r)
Octahedra(s0, s1, s2, s3, s4, s5, s6, r, xc, yc, zc, cva, cvb, cvc)

Defines an octahedra with a central atom.

[s0, s1, s2, s3, s4, s5, s6]: Site names. s0 is the central atom of the octahedra.

[r]: Distance in Å.

[xc, yc, zc]: Parameter names for the coordinates for the central atom.

[cva, cvb, cvc]: Parameter names and values for rotations about the x, y and z axes.

Hexagon_sitting_on_point_in_xy_plane(s1, s2, s3, s4, s5, s6, a)
Hexagon_sitting_on_side_in_xy_plane(s1, s2, s3, s4, s5, s6, a)

Defines a regular hexagon, where the hexagon is sitting on a point or on a side in the x–y plane, respectively.

[s1, s2, s3, s4, s5, s6]: Site names.

[a]: Distance in Å.

Translate(acv, bcv, ccv)
Translate(acv, bcv, ccv, ops)

Performs a translation of the rigid body.

[acv, bcv, ccv]: Amount of the translation in fractional coordinates.

[ops]: Operates on previously defined sites in "ops".

Translate_with_site_start_values(s0, xc, yc, zc)

Performs a translation using the coordinates of s0 as start values.

[s0]: Site name.

[xc, yc, zc]: Parameter names for the coordinates of s0.

Rotate_about_points(cv, a, b)
Rotate_about_points(cv, a, b, pts)

Performs a rotation about a rotation vector specified by two sites.

[cv]: Amount the rigid body is rotated about the specified rotation vector in degrees.

[a, b]: Rotation vector defined by the sites a and b.

[pts]: Operates on previously defined *point_for_site*(s).

Note: Do not include points rotated about in the "operate on points" list of the *Rotate_about_points* macro. For example, in

```
Rotate_about_points(@ 1 0, C1, C2, " C3 C4 C5 C6 ")
```

the points C1 and C2 are not included in the "points operated on" list. Note also that *Rotate_about_points* without a "points operated on" list will operate on all previously defined *point_for_site*(s). Therefore, when an "operate on points" list is not defined then it is necessary to place the "points rotated about" after the *Rotate_about_points* macro. It is best to specify an "operate on points" list when in doubt.

Rotate_about_these_points(cv, a, b, ops)

Performs a rotation about a rotation vector specified by two sites.

[cv]: Amount the rigid body is rotated about the specified rotation vector in degrees.

[a, b]: Rotation vector defined by the sites a and b.

[ops]: Operates on previously defined point_for_site(s).

Rotate_about_axies(cva, cvb)
Rotate_about_axies(cva, cvb, cvc)

Performs a rotation about the axes.

17. MISCELLANOUS

17.1 ... Threading

TOPAS is threaded to a large extent; this allows the utilization of multiple processors resulting in faster program execution. The degree of speedup is computer and problem dependent. For non-trivial problems, the gain is 2 to 4 for a 4-core laptop PC with four i7 processors. Attention is paid to reducing memory usage at the thread level. This is particularly apparent when using rigid bodies or occupancy merge where Version 7 uses far less memory than Version 5. Except for penalties all items are threaded; they include peak generation, all convolutions, all derivatives that are a function of *Ycalc*, equations that are a function of changing variables such as *X*, *Th*, *D_spacing* etc..., Pawley refinement, structure refinement, charge flipping, magnetic refinement, stacking faults, PDF refinement, conjugate gradient solution method and Indexing.

17.1.1 ....Setting the maximum number of threads

The program defaults to using the maximum number of threads available. The user can limit this behaviour by editing the file MAXNUMTHREADS.TXT. This file is read on program startup; it contains a single number, let's call it Max_Threads_File, which defines the maximum number of threads. Non-existence of the file or a Max_Threads_File of zero results in the program using the maximum number of threads available. If Max_Threads_File is negative, then the maximum number of threads is set to the following:

$$\text{Max\_Number\_Threads} = \text{Max}(1, \text{Max\_Threads\_File} + \text{Max\_Threads\_Available});$$

17.2 ... Restraining background using the Bkg_at function

The Chebyshev background function, *bkg*, can sometimes misbehave during Pawley, Le Bail or deconvolution refinements. In the case of *xdd* deconvolution refinements, the *Deconvolution_Bkg_Penalty* stabilizes *bkg* in most cases. In cases of instability, however, the Bkg_at(x) function can be used in penalty functions to guide the shape of *bkg.* Bkg_at(x) returns the value of *bkg* at the x-axis position of x. Here's an example use of Bkg_at as applied to TOF data:

```
bkg  @  0.443519294`  0.0200324829`  0.0113774736`
penalty = 1000000 (Bkg_at(2036)  - 0.43)^2;
penalty = 1000000 (Bkg_at(9000)  - 0.50)^2;
penalty = 1000000 (Bkg_at(14600) - 0.50)^2;
```

The first penalty restrains the value of *bkg* at x=2036 to 0.43. Typically, only two to three Bkg_at penalties are necessary. The values of 0.43, 0.5 and 0.5 can be determined graphically.

17.3 ... Calculation of structure factors

The structure factor F for a reflection (h k l) is the complex quantity:

$$F = \Sigma_s (A_S + i\ B_S) \Sigma_a (f_{o,a} + f_a{}' + i\ f_a{}'')\ O_a \qquad \text{(17-1)}$$

The summation Σ_s is over the sites of the unit cell and the summation Σ_a is over the atoms residing on site s. O_a and $f_{o,a}$ corresponds to the site occupancy and the atomic scattering factor for atom 'a' respectively. f_a' and f_a'' are the anomalous dispersion coefficients for atom 'a'. A_S and B_S corresponds to the cosine and sine summations for site 's', or,

$$A_S = \Sigma_e \, T_{s,e} \cos(2\pi \, \mathbf{h} . \mathbf{r}_e), \quad B_S = \Sigma_e \, T_{s,e} \sin(2\pi \, \mathbf{h} . \mathbf{r}_e) \tag{17-2}$$

where $T_{s,e}$ is the temperature factor and the summation Σ_e is over the equivalent positions of site 's' as dictated by the space group. Defining:

$$f_{o,s} = \Sigma_a \, f_{o,a} \, O_a, \quad f_s' = \Sigma_a \, f_a' \, O_a, \quad f_s'' = \Sigma_a \, f_a'' \, O_a \tag{17-3}$$

and separating the real and imaginary components gives:

$$F = \Sigma_s (A_s + i \, B_s)(f_{o,s} + f_s' + i \, f_s'') \tag{17-4}$$

$$F = \Sigma_s (A_s (f_{o,s} + f_s') - B_s \, f_s'') + i \, \Sigma_s (A_s \, f_s'' + B_s (f_{o,s} + f_s'))$$

$$\text{or, } F = A + i \, B$$

The intensity is proportional to the complex conjugate of the structure factor, or,

$$F^2 = A^2 + B^2 \tag{17-5a}$$

or,

$$F^2 = A_{01}^2 + B_{01}^2 + A_{11}^2 + B_{11}^2 + 2 \, B_{01} \, A_{11} - 2 \, A_{01} \, B_{11} \tag{17-5b}$$

$$\text{where } A_{01} = \Sigma_s \, A_s (f_{o,s} + f_s'), \quad A_{11} = \Sigma_s \, A_s \, f_s''$$

$$B_{01} = \Sigma_s \, B_s (f_{o,s} + f_s'), \quad B_{11} = \Sigma_s \, B_s \, f_s''$$

$$\text{and } A = A_{01} - B_{11}, \quad B = B_{01} + A_{11}$$

Atomic scattering factors, $f_{o,a}$, comprise 11 values per atom and are found in the file ATMSCAT_11.CPP. Correspondingly 9 values per atom, obtained from the International Tables, are found in the file ATMSCAT_9.CPP. Use of either 9 or 11 values can be invoked by running the batch files USE_9F0 and USE_11F0 respectively. Dispersion coefficients, f_a' and f_a'', are by default from http://www.cxro.lbl.gov/optical_constants/asf.html.

These data, found in the SSF directory, covers the energy range from 10 to 30000eV. The use of *use_tube_dispersion_coefficients* forces the use of dispersion coefficients from the International Tables for X-ray Crystallography (1995), Vol.C, p384-391 and 500-502, and for O2- from Hovestreydt (1983). These data are in discrete energy steps corresponding to wavelengths typically found in laboratory X-ray tubes. For neutron diffraction data, $f_a' = f_a'' = 0$ and $f_{o,a}$ is replaced by the bound coherent scattering length (found in the NEUTSCAT.CPP file) for atom 'a' obtained from:

http://www.ccp14.ac.uk/ccp/web-mirrors/neutrons/n-scatter/n-lengths/LIST~1.HTM.

17.3.1 ....Friedel pairs

For centrosymmetric structures the intensities for a Friedel reflection pair are equivalent, or, F^2(h k l) = F^2(-h-k-l). This holds true regardless of the presence of anomalous scattering and regardless of the atomic species present in the unit cell. This equivalence in F^2 is due to $B_{01} = B_{11} = 0$ and thus:

$$F = A_{01} + i\, A_{11} \quad \text{and} \quad F^2 = A_{01}^2 + A_{11}^2 \tag{17-6}$$

For non-centrosymmetric structures and for the case of no anomalous scattering, or for the case where the unit cell comprises a single atomic species, then F^2(h k l) = F^2(-h-k-l). Or, for a single atomic species we have:

$$B_{01}\, A_{11} = (f_0 + f')(\Sigma_S\, B_S)\, f''(\Sigma_S\, A_S), \quad A_{01}\, B_{11} = (f_0 + f')(\Sigma_S\, A_S)\, f''(\Sigma_S\, B_S) \tag{17-7}$$

$$\text{or,}\ B_{01}\, A_{11} = A_{01}\, B_{11}$$

and thus, from cancellation in Eq. (17-5b) we get:

$$F^2(\mathbf{h}) = F^2(-\mathbf{h}) = A_{01}^2 + B_{01}^2 + A_{11}^2 + B_{11}^2 \tag{17-8}$$

For non-centrosymmetric structures and for the case of anomalous scattering and for a structure comprising more than one atomic species then $F^2(\mathbf{h}) \neq F^2(-\mathbf{h})$.

17.3.2 ....Powder data

Friedel pairs are merged for powder diffraction data meaning that the multiplicities as determined by the hkl generator includes the reflections (h k l) and (-h -k -l); this improves computational efficiency. Eq. (17-5b) gives the correct intensity for unmerged Friedel pairs and thus it cannot be used for merged Friedel pairs. Using the fact that:

$$A_{01}(\mathbf{h}) = A_{01}(-\mathbf{h}), \quad A_{11}(\mathbf{h}) = A_{11}(-\mathbf{h}) \tag{17-9}$$

$$B_{01}(\mathbf{h}) = B_{01}(-\mathbf{h}), \quad B_{11}(\mathbf{h}) = B_{11}(-\mathbf{h})$$

then F^2 from Eq. (17-5b) in terms of $B_{01}(\mathbf{h})$ and $B_{11}(\mathbf{h})$ evaluates to:

$$F^2(\mathbf{h}) = Q_1 + Q_2 \tag{17-10}$$

$$F^2(-\mathbf{h}) = Q_1 - Q_2$$

$$\text{where } Q_1 = A_{01}^2 + B_{01}^2 + A_{11}^2 + B_{11}^2$$

$$\text{and } Q_2 = 2\,(B_{01}\, A_{11} - A_{01}\, B_{11})$$

and for merged Friedel pairs we get:

$$F^2(\mathbf{h}) + F^2(-\mathbf{h}) = 2\, Q_1 \tag{17-11}$$

The factor of 2 in Eq. (17-11) is dropped as its included in multiplicity from the hkl generator. Thus, the final equation describing F^2 for powder diffraction data for merged Friedel pairs is:

$$F^2(\mathbf{h})_{merged} = Q_1 \qquad\qquad (17\text{-}12)$$

The reserved parameter names of *A01*, *A11*, *B01* and *B11* can be used to obtain unmerged real, imaginary and F^2 components and the merged F^2. The following macros have been provided in TOPAS.INC:

```
macro F_Real_positive { (A01-B11) }
macro F_Real_negative { (A01+B11) }
macro F_Imaginary_positive { (A11+B01) }
macro F_Imaginary_negative { (A11-B01) }
macro F2_positive { (F_Real_positive^2 + F_Imaginary_positive^2) }
macro F2_negative { (F_Real_negative ^2 + F_Imaginary_negative^2) }
macro F2_Merged { (A01^2 + B01^2 + A11^2 + B11^2) }
```

Note that F2_Merged = (F2_positive + F2_negative) / 2. The reserved parameters *l_no_scale_pks* and *l_after_scale_pks* for *str* phases are equivalent to the following:

   l_no_scale_pks = Get(*scale*) *M* F2_Merged

   l_after_scale_pks = Get(all_scale_pks) Get(*scale*) *M* F2_Merged

In addition, the macros *Out_F2_Details* and *Out_A01_A11_B01_B11* can be used to output F^2 details.

17.3.3 ....Single crystal data

SHELX HKL4 single crystal data comprise unmerged equivalent reflections and thus Eq. (17-5b) is used for calculating F^2. Equivalent reflections are merged by default and can be unmerged using *dont_merge_equivalent_reflections*. For centrosymmetric structures, merging includes the merging of Friedel pairs and thus Eq. (17-12) is used for calculating F^2. For non-centrosymmetric structures, merging excludes the merging of Friedel pairs and thus (17-5b) is used for calculating F^2. *dont_merge_Friedel_pairs* prevent merging of Friedel pairs. *ignore_differences_in_Friedel_pairs* force the use of Eq. (17-12) for calculating F^2. The reserved parameter name *Mobs* returns the number of observed reflections belonging to a family of reflections.

Merging of equivalent reflections reduces the computational effort and is useful in the initial stages of structure refinement. Only a single intensity is calculated for a set of equivalent reflections even in the absence of merging. Thus, equivalent reflections and Friedel pairs are remembered and intensities appropriated as required.

*.SCR data is typically generated from a powder pattern and comprises merged equivalent reflections including merged Friedel pairs. Consequently, Eq. (17-12) is used for calculating F^2; definitions of *dont_merge_equivalent_reflections*, *dont_merge_Friedel_pairs* and *ignore_differences_in_Friedel_pairs* are ignored.

17.3.4 ....The Flack parameter

[*Flack* E]

For single crystal data and for non-centrosymmetric structures the *Flack* parameter (Flack, 1983) scales $F^2(\mathbf{h})$ and $F^2(-\mathbf{h})$ as follows (see the test example YLIDMA.INP):

$$F^2(\mathbf{h}) = Q_1 + (1 - 2 \text{ Flack}) Q_2 \qquad\qquad (17\text{-}13)$$

$$F^2(-\mathbf{h}) = Q_1 - (1 - 2 \text{ Flack}) Q_2$$

17.3.5 ....Single Crystal Output

The macro *Out_Single_Crystal_Details*, see TOPAS.INC, outputs details for single crystal refine-ment, see test example YLIDMA.INP. *Mobs* corresponds to the number of observed reflections be-longing to a family of planes. When Friedel Pairs are not merged then *Mobs* for **h** and –**h** will be different. Phase symmetry is considered in the values for A01, B01, A11 and B11.

17.4 ... Convolution

17.4.1 ....Instrument and sample convolutions

Diffractometer instrument and sample aberration functions used in peak profile synthesis are generated from generic convolutions. For example, the 'simple' axial divergence model is de-scribed using the generic convolution *circles_conv* as defined in the *Simple_Axial_Model* macro. Table 17-1 lists some of the instrument convolutions supported. In addition, the full axial diver-gence model of Cheary & Coelho (1998a, 1998b) is supported.

Table 17-1. Instrument and sample aberration functions in terms of $\varepsilon = 2\theta - 2\theta_k$, where 2θ is the measured angle and $2\theta_k$ the Bragg angle. R_P and R_S correspond to the primary and secondary radius of the diffractometer respectively. ε_m in $^\circ 2\theta$

Aberrations	Name	Aberration function Fn(ε)
Instrument		
Equatorial divergence (fixed divergence slits)	EDFA [°]	$Fn(\varepsilon) = (4\varepsilon_m\varepsilon)^{-\frac{1}{2}}$ $for\ \varepsilon = 0\ to\ \varepsilon_m = -\left(\dfrac{\pi}{360}\right)cot(\theta_k)\,EDFA^2$
Equatorial divergence (variable divergence slits)	EDFL (mm)	$Fn(\varepsilon) = (4\varepsilon_m\varepsilon)^{-\frac{1}{2}}$ $for\ \varepsilon = 0\ to\ \varepsilon_m = -\dfrac{EDFL^2\,sin(2\theta_k)\left(\frac{180}{\pi}\right)}{4R_S^2}$
Size of source in the equatorial plane	TA (mm)	$Fn(\varepsilon) = HatShape,\qquad for -\dfrac{\varepsilon_m}{2} < \varepsilon < \dfrac{\varepsilon_m}{2}$ $where\ \varepsilon_m = \dfrac{\left(\frac{180}{\pi}\right)TA}{R_S}$
Specimen tilt; thickness of sample surface as pro-jected onto the equatorial plane	ST (mm)	$Fn(\varepsilon) = HatShape,\qquad for -\dfrac{\varepsilon_m}{2} < \varepsilon < \dfrac{\varepsilon_m}{2}$ $Where\ \varepsilon_m = \dfrac{\left(\frac{180}{\pi}\right)cos(\theta_k)ST}{R_S}$
Receiving slit length in the axial plane	SL (mm)	$Fn(\varepsilon) = \left(\dfrac{1}{\varepsilon_m}\right)\left(1 - \sqrt{\dfrac{\varepsilon_m}{\varepsilon}}\right)$

		$$for\ \varepsilon = 0\ to\ \varepsilon_m = -\left(\frac{90}{\pi}\right)\left(\frac{SL}{R_S}\right)^2 \cot(2\theta_k)$$
Width of the receiving slit in the equatorial plane	SW (mm)	$$Fn(\varepsilon) = HatShape, \qquad for -\frac{\varepsilon_m}{2} < \varepsilon < \frac{\varepsilon_m}{2}$$ $$where\ \varepsilon_m = \frac{\left(\frac{180}{\pi}\right)SW}{R_S}$$
Sample		
Linear absorption coefficient	AB (cm⁻¹)	$$Fn(\varepsilon) = \left(\frac{1}{\delta}\right) Exp\left(-\frac{\varepsilon}{\delta}\right)$$ $$for\ \varepsilon \leq 0\ and\ \delta = 900 \sin(2\theta_k)/(\pi\ AB\ R_S)$$

17.4.2 ....Convolutions in general

TOPAS performs convolution in various ways and the terms "FFT convolution" (Fast Fourier Transform) or "direct convolution" are simplifications. Typically, convolutions are broken down into double summations that can be calculated either directly or by using an FFT. The program uses the method that is fastest as determined by calculating the number of operations required by each method.

Response functions that are known to the program are treated analytically. Response functions that are unknown to the program (such as user defined convolutions) are treated as straight-line segments. Convolution therefore can be i) between two sets of line segments ii) one set of line segments and an analytical expression or iii) simply done analytically. For a response function with N_r data points and a peak comprising N_p points and when straight-line segments are used, the extra cost of the piece wise integration is approximately $3(N_r+N_p)$ operations; this is a small number of operations and the benefit is a high degree of accuracy. Apart from *lor_fwhm* and *gauss_fwhm*, the convolutions described below have discontinuities in 2Th space; associated Fourier transforms are therefore difficult to describe and hence convolution is performed in 2Th space. Response functions that are treated as line segments are:

> *user_defined_convolution*, *capillary_diameter_mm*, *lpsd_th2_angular_range_degrees*

Response functions that are analytically convoluted with line segments are:

> *exp_conv_const*, *hat*, *stacked_hats_conv*

Response functions that comprise a mixture of analytical and straight-line segments are:

> *axial_conv*, *one_on_x_conv*, *circles_conv*

lor_fwhm and *gauss_fwhm* convolutions are convoluted analytically with the emission profile to form the base profile. Convolutions are calculated with an x-axis step size of:

> *Peak_Calculation_Step* = *x_calculation_step* / *convolution_step*

For efficiency *x_calculation_step* should not be defined for data with equal x-axis steps; instead *rebin_with_dx_of* should be used. The following response functions are calculated at smaller step sizes without changing *Peak_Calculation_Step* or N_r:

axial_conv	: Step = *Peak_Calculation_Step* / 2
lpsd_th2_angular_range_degrees	: Step = *Peak_Calculation_Step* / 3
capillary_diameter_mm	: Step = *Peak_Calculation_Step* / 1 to 3

In this manner a high degree of accuracy is maintained and N_p*N_r is left unchanged. Typically, a laboratory diffraction pattern can be accurately synthesized with a *Peak_Calculation_Step* of 0.02 degrees 2Th. The next step to increasing accuracy would be to increase *convolution_step* to 2 and so on. When direct convolution is used then convolution scales by a factor of ($N_r * N_p$). Convolutions that scale by (N_r+N_p) are very fast and are:

> *exp_conv_const*, *hat*, *stacked_hats_conv*

Calculating derivatives of parameters that are a function of a convolution can be demanding. Most convolutions however that have multiple dependent parameters require only one recalculation of the convolution; exceptions are *ft_conv*, *WPPM_ft_conv* and *user_defined_convolution*. In the case of convolutions that comprise multiple convolution parameters, for example, *axial_conv* with its convolution parameters of *primary_soller_angle* etc…, then a recalculation for each of the convolution parameters is required. The following is an overview of convolutions and associated aberrations:

axial_conv	Full Axial divergence model
one_on_x_conv	Equatorial Divergence
circles_conv	Simple axial model
capillary_diameter_mm	Capillary sample
lpsd_th2_angular_range_degrees	LPSD detector
exp_conv_const	Sample penetration
hat	Receiving slit width, sample tilt
stacked_hats_conv	Tube tails

17.4.3 .... Capillary convolution for a focusing convergent beam

The capillary convolution has been extended to include a focusing convergent beam (Coelho & Rowles, 2017); syntax is as follows:

[*capillary_diameter_mm* E]
    capillary_u_cm_inv E
    [*capillary_convergent_beam*] [*capillary_divergent_beam*] [*capillary_parallel_beam*]
    [*capillary_focal_length_mm* E]
    [*capillary_xy_n* #]

See examples LAB6-STOE.INP and LAB6-D8.INP in the directory TEST_EXAMPLES\CAPILLARY. If using a *str* phase then *capillary_u_cm_inv* can be set to the calculated linear absorption coefficient multiplied by a packing density, for example:

```
    prm packing_density 0.31208
    capillary_diameter_mm @ 0.57313
        capillary_u_cm_inv
            = Get(mixture_MAC) Get(mixture_density_g_on_cm3) packing_density;
        capillary_focal_length_mm @ 197.89657
        capillary_convergent_beam
```

If *capillary_focal_length_mm* is not defined, then it defaults to the diffractometer radius *Rs*.

17.4.4 ....ft_conv

[*ft_conv_re_im*] ...
    [*ft_conv_re* E]
    [*ft_conv_im* E]
    [*ft_min* !E]
    [*ft_x_axis_range* !E]
    ' Get(ft_0)
    ' *FT_Break*

Fourier Transform (FT) of a response function that is convoluted into phase peaks using a Fast Fourier Transform (FFT); for example, to convolute a Voigt into a phase the following can be used:

```
ft_conv = Exp(-(Pi FT_K gfwhm)^2 / (4 Ln(2)) - Pi FT_K lfwhm);
    ft_min = 1e-8; ' this is the default; ft_min is optional
    ft_x_axis_range = 40 lfwhm;
```

ft_conv is equal to the two keywords [*ft_conv_re_im ft_conv_re*]. More than one transform can be defined. See *ft_conv* and *WPPM_ft_conv* for details. Here the convolution theorem is used by multiplying the FT of a Gaussian by the FT of a Lorentzian. If the Fourier transforms are separately defined, the program will internally use the convolution theorem. *FT_K* is a reserved parameter name and it returns the transform k divided by the x-axis range of the peak; this range includes *ft_x_axis_range*. *ft_x_axis_range* can be an equation that needs to be set such that the transform decays to near zero; peak tails will otherwise be incorrect. A Lorentzian for example needs a large *ft_x_axis_range* for accurate tails. *ft_min* defines the smallest value to which the transform is calculated to. For example, an already broadened peak in x-axis space will have a relatively narrow transform; the calculation of the transform is therefore terminated when FT(k)/FT(k=0)<*ft_min*. Transform values for larger k are then set to zero. If(,,) constructs can instead be used within the transform equation for further control; for example:

```
ft_conv = If (FT_K > D, FT_Break, Sphere(FT_K, D));
```

Here the calculation of the FT is terminated when *FT_K*>D using *FT_Break*. Get(ft_0) returns FT(k=0) and can be used within the *ft_conv* equation, for example:

```
ft_conv = {
    def a = Exp(-Pi FT_K lf);
    return If(a < 1e-6 Get(ft_0), FT_Break, a);
}
```

ft_conv integrates with convolutions that are performed in direct space. It can be used within peak stack operations and it can be a function of the reserved parameter names:

>   *H*, *K*, *L*, *M*, *Th*, *Xo*, *D_spacing*, *FT_K*

Multiple *ft_conv* (s) can be defined at either the *xdd* or phase level. When defined at the *xdd* level the convolution is applied to all phases of that *xdd*. The TEST_EXAMPLES\FT directory comprises examples that use *ft_conv*. For a typical Rietveld refinement, an *ft_conv* used to describe a Voigt is almost as fast as the analytical equivalent as seen in example FT\ALVO4A.INP. For high accuracy the range of the peak, as defined with *ft_x_axis_range*, needs to be large, up to 400 FWHM for a Lorentzian; in these cases, the *ft_conv* is considerably slower as seen in FT\VOIGT.INP.

FT\ALVO4A.INP compares the use of *spherical_harmonics_hkl* with and without ft_conv as follows.

```
prm csl 50 min 3 max = Min(Val 2 + 0.1, 10000);
prm csg 50 min 3 max = Min(Val 2 + 0.1, 10000);
prm csl_fwhm = 0.1 Rad Lam / (csl Cos(Th));
prm csg_fwhm = 0.1 Rad Lam / (csg Cos(Th));
if 1 {
    ' Spherical Harmonics
    spherical_harmonics_hkl sh
        sh_order 2
        load sh_Cij_prm {
            y00    !sh_c00  1
            y20    sh_c20   0
            y21p   sh_c21p  0
            y21m   sh_c21m  0
            y22p   sh_c22p  0
            y22m   sh_c22m  0
        }
    existing_prm csl_fwhm *= sh;
    existing_prm csg_fwhm *= sh;
}
if 0 {
    ' use analytical Lorentzian and Gaussian convolution
    lor_fwhm = csl_fwhm;
    gauss_fwhm = csg_fwhm;
} else {
    ' use Fourier Transform convolution
    ft_conv = Exp(-(Pi FT_K csg_fwhm)^2 / (4 Ln(2)) - Pi FT_K csl_fwhm);
        ft_x_axis_range = 45 csl_fwhm + 4 csg_fwhm;
}
```

The speed of the analytical convolution is greater not simply because describing the peak analytically is faster but because derivatives of multiple parameters for *lor_fwhm* (or *gauss_fwhm*) requires only one peak calculation; whereas for *ft_conv* the peak is recalculated for each independent parameter that it is a function of.

17.4.4.1...... ft_conv compared to user_defined_convolution

If a response function is known in x-axis space then it is often best to perform the convolution in x-axis space rather that describing the FT of the response function using *ft_conv*.

user_defined_convolution can be used to perform convolution in x-axis space and the speed at which it operates is as fast or faster than *ft_conv* depending on the x-axis range of the response function; this is demonstrated in FT\LORENTZIAN.INP. For each peak, *user_defined_convolution* estimates the computational effort required to perform the convolution eiether directly or with an FFT and chooses the one with the least computational effort. Examples that use *user_defined_convolution* are as follows:

```
FT\LORENTZIAN.INP
TOF\TOF_BANK2_2.INP
WPPM\GAMMA.INP
UDEFA.INP
```

UDEFA.INP shows how to convolute a function with discontinuities; i.e.

```
user_defined_convolution = Exp(-20 X^2); min = -.2; max = .5;
```

The FT for functions with such discontinuities often cannot be described analytically.

17.4.4.2..... FFT versus direct summation

Typically an FFT convolution for a response function that comprise histograms is quoted as comprising $O(N \log_2 N)$ operations (Cooley–Tukey algorithm for example). Direct convolution is quoted as comprising $O(N^2)$ operations and is only faster for $N<128$. However, in XRD work a direct convolution rather than an FFT working on real numbers is often faster for $N \sim< 256$ to 512 as the comparison of the $O(N \log_2 N)$ versus $O(N^2)$ is not strictly correct. To see why consider a response function comprising 3 points and a peak comprising 5 points. A convolution can be pictured as the response function R moving along the peak P as follows:

```
P    0 0 0 1 1 1 1 1 0 0 0
R       - - x
R         - x x
R           x x x
R             x x x
R               x x x
R                 x x -
R                   x - -
```

In this representation each 'x' can be considered a multiply; in direct convolution this makes a total of 15 multiplies ($N_r * N_p$) and not N^2 where $N/2 \leq (N_r + N_p) \leq N$. To perform such a convolution using an FFT, the number of operations is approximately $4*16*\log_2 16=256$ multiplies where 16 is the closest power of 2 to $N_r + N_p$. Of course, FFT routines typically also have special cases for small N; nonetheless $N=256$ to 512 is not small and many peaks in XRD work typically comprise less points and many of the response functions have a small N_r; these include axial divergence, equatorial divergence, receiving slit width, capillary convolution, LPSD convolution and often sample penetration. Another factor favoring direct convolution for modest N_r and N_p is the fact that modern processors such as the Intel i7 are very fast when data in cache memory are arranged sequentially and accessed sequentially. In fact, non-sequential operations can be as much as 8 times slower than for the sequential case.

17.4.5 ....WPPM

[*WPPM_ft_conv_re_im* E]...
    [*WPPM_ft_conv_re* E]
    [*WPPM_ft_conv_im* E]
    WPPM_L_max E
    WPPM_th2_range E
    [*WPPM_break_on_small* !E]
    [*WPPM_correct_ls*]

Examples referred to in this section reside in the TEST_EXAMPLES\WPPM directory. *WPPM_ft_conv* is equal to [*WPPM_ft_conv_re_im WPPM_ft_conv_re*].

17.4.5.1...... WPPM in 2Th space

The WPPM microstructure analysis (Scardi & Leoni, 2001; Leoni *et al*. 2004; David *et al*. 2010) for domains comprising spheres and a gamma distribution can be implemented using *user_defined_convolution* operating in 2Th space as shown in GAMMA.INP.

17.4.5.2...... WPPM using fit_obj(s)

For cases where microstructure broadening is far greater than instrument/emission profile broadening then *fit_obj*'s can be used to describe the peak shape (see GAMMA-FIT-OBJ.INP and SPHERE-FIT-OBJ.INP), for example:

```
fn gamma_mu_variance(mu, v, xo) {
   def s = 2 ( Sin( X Pi/360) - Sin(xo Pi/360) ) / lam;
   def p0 = Pi s mu;
   def p = If(Abs(p0) < 1e-10, 1, p0);
   def q = 2 p / v;
   return mu v / p^4
      (
         2 p^2 / (2 + v) + (v/(2+ 3 v + v^2)) (1 - (1 + q^2)^(-.5 v)
         Cos(v ArcTan(q)) - 2 p (1 + q^2)^(-.5 (v+1)) Sin( (1 + v)
         ArcTan(q)))
      );
}
```

Example SUPER-LORENTZIAN.INP is useful for asking the question; can spheres with a gamma distribution describe a $1/(1+x^2)^m$ type function? Example COMPARE-1.INP is useful for asking the question; can a Voigt fit to a particular case of spheres with a gamma distribution?

17.4.5.3...... WPPM using WPPM_ft_conv

WPPM_ft_conv describes a FT in *s* space and performs a convolution on phase peaks that have been interpolated to *s* space, for example:

```
    WPPM_ft_conv = 1 - 1.5 WPPM_L / D + 0.5 (WPPM_L / D)^3;
        WPPM_L_max = D;
        WPPM_th2_range = 25 .1 Rad Lam / (D Cos(Th));
        WPPM_break_on_small 1e-7
        WPPM_correct_Is
```

The result is then interpolated back to 2Th space. Interpolations are scaled such that I(*s*)*ds* = I(θ)*d*θ when *WPPM_correct_ls* is defined; the effects of this scaling is typically small at low angles and becomes noticeable at very high angles reaching a maximum at 180 degrees 2Th where the derivative of Cos(*Th*) is at a maximum.

When multiple *WPPM_ft_conv*(s) are defined then the program will internally use the convolution theorem.

WPPL_L is a reserved parameter name that returns the transform parameter.

WPPM_L_max defines the maximum *WPPL_L*.

Get(ft_0) and FT_Break can both be used in *WPPM_ft_conv* in a manner similar-to *ft_conv*.

The calculation of the Fourier transform is terminated when *WPPM_ft_conv_re* is less than *WPPM_break_on_small* multiplied by the value of *WPPM_ft_conv_re* evaluated at WPPM_L = 0. If *WPPM_break_on_small* is not defined then no check is made to terminate the transform.

The tails of WPPM peaks extend for almost the whole diffraction pattern; they can be shortened using *WPPM_th2_range*; in the above example this range has been written in terms of the *fwhm* as defined in the Scherrer equation. *WPPM_ft_conv* can be a function of the following reserved parameter names:

       H, K, L, M, Th, Xo, D_spacing, WPPM_L

Example S-SPHERE-1.INP uses *WPPM_ft_conv* to fit to a synthesized WPPM generated peak with identical results. Example CUBE-LN-NORMAL-1.INP can be used to test these macros. Lattice parameters appearing within the macros are made constant using Constant; these convolutions are therefore made independent of lattice parameter changes and hence separate convolutions are not initiated whilst calculating lattice parameter derivatives.

WPPM_Ln_k is a reserved parameter name that returns Ln of an integer and is used to calculate Ln(Kc *WPPM_L*) in a fast manner.

The example LN-NORMAL-1.INP can be used for visualizing a Ln normal distribution. It uses the Ln_Normal_x_at_CD function to determine the limit of the distribution.

17.4.6 ....Microstructure convolutions

The Double-Voigt approach (e.g. Balzar, 1999) is supported for modeling microstructure effects. Crystallite size and strain comprise Lorentzian and Gaussian component convolutions varying in 2θ as a function of 1/cos(θ) and tan(θ) respectively.

17.4.6.1...... Preliminary equations

The following preliminary equations are based on the unit area Gaussian, GUA(x), Lorentzian, LUA(x), and pseudo-Voigt PVUA(x) functions as given in Table 5-2.

Height of $G_{UA}(x)$ and $L_{UA}(x)$ respectively:

$$G_{UAH} = G_{UA}(x=0) = g_1 / fwhm$$
$$L_{UAH} = L_{UA}(x=0) = l_1 / fwhm$$

Gaussian and Lorentzian respectively with area A:

$$G(x) = A\, G_{UA}(x)$$
$$L(x) = A\, L_{UA}(x)$$

Height of G(x) and L(x) respectively:

$$G_H = A\, G_{UA}$$
$$L_H = A\, L_{UAH}$$

Integral breadth of Gaussian and Lorentzian respectively:

$$\beta_G = A / G_H = 1 / G_{UAH} = fwhm / g_1$$
$$\beta_L = A / L_H = 1 / L_{UAH} = fwhm / l_1$$

Unit area Pseudo Voigt, PV_{UA}:

$$PV_{UAH} = \eta\, L_{UAH} + (1-\eta)\, G_{UAH}$$
$$\beta_{PV} = 1 / PV_{UAH}$$

A Voigt is the result of a Gaussian convoluted by a Lorentzian:

$$V = G(fwhm_G) \otimes L(fwhm_L)$$

where "$\otimes$" denotes convolution and fwhmG and fwhmL are the FWHM of the Gaussian and Lorentzian components. A Voigt can be approximated using a Pseudo Voigt. This is done numerically where:

$$V(x) = G(fwhm_G) \otimes L(fwhm_L) = PV_{UA}(x, fwhm_{PV})$$

By changing units to s (Å^{-1}):

$$s = 1/d = 2\sin(\theta) / \lambda$$

and differentiating and approximating $ds/d\theta = \Delta s / \Delta\theta$ we get:

$$\Delta s = (2\cos(\theta) / \lambda)\, \Delta\theta$$

thus:

$$fwhm(s) = fwhm(2\theta)\cos(\theta) / \lambda$$
$$IB(s) = IB(2\theta)\cos(\theta) / \lambda$$

17.4.6.2...... Crystallite size and strain

Crystallite Size: Gaussian and Lorentzian component convolutions are:

$$\text{fwhm}(2\theta) \text{ of Gaussian} = (180/\pi)\,\lambda\,/\,(\cos(\theta)\,CS\_G)$$

$$\text{fwhm}(2\theta) \text{ of Lorentzian} = (180/\pi)\,\lambda\,/\,(\cos(\theta)\,CS\_L)$$

$$\beta(2\theta) \text{ of Gaussian} = (180/\pi)\,\lambda\,/\,(\cos(\theta)\,CS\_G\,g_1)$$

$$\beta(2\theta) \text{ of Lorentzian} = (180/\pi)\,\lambda\,/\,(\cos(\theta)\,CS\_L\,l_1)$$

or, according to Balzar (1999), in terms of s, βGS and βCS:

$$\text{fwhm}(s) \text{ of Gaussian} = (180/\pi)\,/\,CS\_G$$

$$\text{fwhm}(s) \text{ of Lorentzian} = (180/\pi)\,/\,CS\_L$$

$$\beta_{GS}(s) = \beta(s) \text{ of Gaussian} = (180/\pi)\,/\,(CS\_G\,g_1)$$

$$\beta_{CS}(s) = \beta(s) \text{ of Lorentzian} = (180/\pi)\,/\,(CS\_L\,l_1)$$

The macros *CS_L* and *CS_G* are used for calculating the CS_L and CS_G parameters respectively. Determination of the volume weighted mean column height LVol, LVol-IB and LVol-FWHM is as follows:

$$\text{LVol-IB} = k\,/\,\text{Voigt\_Integral\_Breadth\_GL}\,(1/CS\_G, 1/CS\_L)$$

$$\text{LVol-FWHM} = k\,/\,\text{Voigt\_FWHM}(1/CS\_G, 1/CS\_L)$$

The macro *LVol_FWHM_CS_G_L* is used for calculating LVol-IB and LVol-FWHM.

Strain: Strain_G and Strain_L parameters corresponds to the fwhm(2θ) of a Gaussian and a Lorentzian that is convoluted into the peak, or,

$$\text{fwhm}(2\theta) \text{ of Gaussian} = \text{Strain\_G}\,\tan(\theta)$$

$$\text{fwhm}(2\theta) \text{ of Lorentzian} = \text{Strain\_L}\,\tan(\theta)$$

$$\beta(2\theta) \text{ of Gaussian} = \text{Strain\_G}\,\tan(\theta)\,/\,g_1$$

$$\beta(2\theta) \text{ of Lorentzian} = \text{Strain\_L}\,\tan(\theta)\,/\,l_1$$

or, according to Balzar (1999), in terms of s, β_{CD} and β_{GD}:

$$\text{fwhm}(s) \text{ of Gaussian} = \text{Strain\_G}\,\sin(\theta)\,/\,\lambda = \text{Strain\_G}\,s\,/\,2$$

$$\text{fwhm}(s) \text{ of Lorentzian} = \text{Strain\_L}\,\sin(\theta)\,/\,\lambda = \text{Strain\_L}\,s\,/\,2$$

$$\beta_{GD}(s)/s_0\,s = \beta(s) \text{ of Gaussian} = (\text{Strain\_G}\,/\,g_1)\,s\,/\,2$$

$$\beta_{CD}(s)/s_0\,s = \beta(s) \text{ of Lorentzian} = (\text{Strain\_L}\,/\,l_1)\,s\,/\,2$$

The macros *Strain_L* and *Strain_G* are used for calculating the Strain_L and Strain_G parameters respectively. From these equations we get:

$$\beta_{GD}(s) = s_0\,\text{Strain\_G}\,/\,(2\,g_1)$$

$$\beta_{CD}(s) = s_0\,\text{Strain\_L}\,/\,(2\,l_1)$$

According to Balzar (1999), equation (34):

$$e = \beta_D(2\theta)\,/\,(4\,\tan(\theta))$$

where $\beta_D(2\theta)$ is the fwhm of a Voigt comprising a Gaussian with a fwhm = Strain_G Tan(θ) and a Lorentzian with a fwhm = Strain_L Tan(θ). The value for e0 is given by:

$$4 \text{ e0 Tan}(\theta)= \text{FWHM of the Voigt from Strain\_L and Strain\_G}$$
$$= \text{Voigt\_FWHM(Strain\_L, Strain\_G) Tan}(\theta)$$

or,

$$\text{e0} = \text{Voigt\_FWHM(Strain\_L, Strain\_G)}(\pi / 360) / 4$$

The macro *e0_from_Strain* calculates e0 using the equation function Voigt_FWHM_GL.

17.5 ... Loading of INP files

17.5.1 ....if {} else if {} else {}

'if' operates during the loading of pre-processed INP files, syntax is as follows (see TEST_EXAMPLES\ZRO2.INP):

```
if expression {
} else if expression {
} else expression {
}
```

expression can be any valid TOPAS equation without the semicolon; in addition, *expression* can contain the functions Prm_There(prm_name) and Obj_There(obj_name). The following is equivalent to a /* */ block comment:

```
if 0 {
    ...
}
```

A more complex construct could look something like:

```
xdd
    local aaa 1
    str ...
        local aaa 2
    str ...
        local aaa 3
    hkl_Is
        if Prm_There(aaa) {
            Out(aaa, "\nThis is the aaa at the xdd level %-1.6f")
            if aaa == 2 {
                Out_String("\nNot written to file as aaa at the xdd level is 1")
            }
        } else if Obj_There(hkl_Is) {
            Out_String("\nYes this is a hkl_Is phase")
        } else {
            Out_String("\naaa is not there and this is not a hkl_Is phase")
        }
    for xdds {
        if And(Obj_There(neutron), Obj_There(pk_xo)) {
            ' Neutron TOF
```

```
    } }
```

17.6 ... Functions – fn, def, return, noinline

Functions can be defined using *fn*; here's an example of a recursive function:

```
fn factorial(x) { return If(x == 1, 1, x factorial(x-1)); }
prm = factorial(5); : 120
```

There's also the simple form where the *return* statement is implied:

```
fn factorial(x) = If(x == 1, 1, x factorial(x-1));
```

The equation part of *prm* objects can have a function body (see the macro *Robust_Refinement* in TOPAS.INC), for example:

```
prm = { def a = 2; return a; }
```

Most importantly, functions can reference parameters defined using *prm*; this simplifies the writing of *prm* equations and additionally memory usage can be greatly reduced when *noinline* is used. Equations called *def* objects can be used and defined within non-simple functions. Here's an example:

```
fn gauss(a, x, f, g) {
    def a1 = 2 Sqrt(Ln(2) / Pi) / f;
    def a2 = 4 Ln(2);
    def a3 = (x / f);
    return a1 Exp(-a2 a3^2);
}
```

A *def* object must be defined prior to its use. They can be assigned to other *def* objects but not to objects of *prm* type. In other words, *prm* objects are write-protected within functions. The arguments to functions can be *def* or *prm* objects. c-style braces can be used to scope variables; the following will throw an exception due to the attempted use of an uninitialized *def* object:

```
fn foo(x) { def a; { def a = x; } return a; }
prm = foo(3); : 0 ' Exception thrown
```

The following will not throw an exception as the simplification routines recognizes '0':

```
Fn a(x) = x undefined_name 0; prm = a(3); : 0
```

Functions can be nested; for example:

```
fn foo() {
    def a, b;
    a = 3; b = 2;
    fn nested(x, y) { return Sqrt(x^2 + y^2); }
    return nested(a, b);
}
prm = foo(); : 3.60555
```

def and *prm* objects have scope which determines the actual object used.

Here *def* 'a' is returned:

```
fn a(a) { def a = 2; return a; } prm = a(1) : 2
```

Here *prm* 'a' is returned:

```
prm a = 2; fn a() = a; prm = a(); : 2
```

Here the argument 'a' is returned:

```
prm a = 2; fn a(a) = a; prm = a(3); : 3
```

Function specifics:

- *fn*'s are a kernel operation and not a pre-processor operation.
- *fn*'s must be defined prior to their use.
- *fn* arguments are optional but parentheses must be used.
- a *fn* cannot be defined with a name of a previously defined *fn* name.
- *fn*'s are inlined by default.
- Non-nested *fn*'s can be prevented from being inlined with the *noinline* prefix.
- nested functions cannot be prefixed with *noinline*.

Use of *noinline* can often be slower than not using *noinline*; this is because a stack mechanism is used for the *fn* arguments, additionally the global simplification routines cannot simplify what's inside a *noinline* function. Functions are therefore 'inlined' (the word 'expand' is sometimes used) by default. A macro can be considered an inlined function and there's no difference in the how the following is finally processed:

```
fn my_max(a, b, c) = Max(a, b, c);
macro & my_max(& a, & b, & c) { Max(a, b, c) }
```

The macro, by definition, is inlined in the pre-processed INP file. In the case of *fn*, the program will inline 'my_max'. Prefixing *fn* with *noinline* prevents inlining, for example:

```
noinline fn gauss(x, f)=(2 Sqrt(Ln(2)/Pi)/f) Exp(-4 Ln(2)((X-x)/f)^2);
```

Its best to inline small functions as it gives the simplification routines a chance to simplify what's inside the function with regards to its surroundings. Consider the following:

```
noinline fn a(b, c) = b^2 + c^2;
prm  p1 1
prm !p2 1
prm  p3 1
prm !p4 1
prm  p5 = a(p1, p2) + a(p3, p4); : 0
```

Without inlining, the simplification routines won't see that p2 and p4 are constants inside the 'a' function and hence no simplification is performed; the 'a' function will be called twice, and the stack used twice. Note, stack here refers to the computer algebra stack. With inlining, p5 after simplification reduces to:

```
prm p5 = p1^2 + p3^2 + 2; : 0
```

In the case of large functions, not inlining may increase performance as the signaling of equation nodes for recalculation will be reduced. Inlined functions have scope allowing the use of the Get(...) function, for example:

```
fn lat(h, k, l) = h Get(a) + k Get(b) + l Get(c);
str ...
    lor_fwhm = lat(H, K, L) - lat(-H, -K, -L);
```

17.6.1 ....Subject independent single crystal refinement

The example \FUNCTIONS\ALVO4-FN.INP performs a single crystal refinement using computer algebra. No subject dependent keywords are used and instead only the following six keywords are used:

fn, *noinline*, *def*, *return*, *prm*, *restraint*

The speed of ALVO4-FN.INP is 7.4 times slower than the comparable subject dependent equivalent of ALVO4-NORMAL.INP. Much of the difference in speed is in the calculation of the Cosines necessary for the structure factors. Importantly convergence and the behavior of the parameters are similar. The placement of noinline is important. Also used is *out_refinement_stats* which outputs the following:

```
First pass equation statistics excluding attribute equations
    Number of equations        : 534
    Number of nodes            : 99751
    Number of nodes if expanded : 12070283

Number of penalties/restraints: 532
Number of independent penalty/restraints parameters: 58
Number of penalties/restraints: 532
Number of independent penalty/restraints parameters: 58

Time    0.13
Second pass equation statistics excluding attribute equations
    Before/After equation simplification
      Number of equations        : 549   553
      Number of nodes            : 99766  8354
      Number of nodes if expanded : 12070298  228183
Number of objects taking part in refinement: 73
Number of dependent parameters with derivatives wrt to Ycalc: 15
```

The ALVO4-FN.INP demonstrates the ease at which an entire single crystal refinement can be performed; it should allow for user defined temperature factors etc.

17.6.2 ....Computer algebra and out_refinement_stats

The computer algebra system CAS in version 5 (Coelho *et al.*, 2011) is around 2 to 4 times faster than version 4; compare with running ROSENBROCK-10.INP or PVS.INP. The CAS has been re-worked and it now operates on a global level where equations are simplified across all objects. The *out_refinement_stats* keywords, for SERINE_I_EVANS_N_TA_BANG_ROT.INP for example, out-puts the following equation statistics:

```
Second pass equation statistics excluding attribute equations
   Before/After equation simplification
      Number of equations        : 2707  3085
      Number of nodes            : 22941  16671
      Number of nodes if expanded : 1706390373  1070170132


Number of objects taking part in refinement: 2595
Number of dependent parameters with derivatives wrt to Ycalc: 2319
```

17.7 ... CIF

The following macros and Get's can be used to output data in CIF format:

```
Out_CIF_STR(file)
Out_CIF_ADPs(file)
Out_CIF_STR(file, with_id)
Out_CIF_Bonds_Angles(file)
Get(number_of_parameters)
Get(refine_ls_shift_on_su_max)
Get(weighting)
Xi = a reserved parameter name
```

_refine_ls_shift / su_max can be accessed using Get(refine_ls_shift_on_su_max) when *do_errors* is defined and when *continue_after_convergence* is NOT defined. A message similar-to the fol-lowing is displayed on calculation:

```
refine_ls_shift_on_su_max 0.409610469 corresponds to parameter m501b939c_3 of object
prm_10
```

Get(*weighting*) and *Xi* can be used as follows:

```
    xdd_out file append load out_record out_fmt out_eqn {
      " %9.0f" = Xi;
      " %11.5f" = X;
      " %11.5f" = Ycalc;
      " %11.5f" = Yobs;
      " %11.5f\n" = Get(weighting);
    }
```

Get(*weighting*) returns *weighting* as defined by the User; if *weighting* is not defined then the fol-lowing is returned:

1 / Max(1, *Yobs*),   if *SigmaYobs* does not exist

1 / *SigmaYobs*^2,   if *SigmaYobs* does exist

Get(*weighting*) returns zero for x-axis regions that are excluded using *exclude*. If *weighting* is a function of *Ycalc* etc... then it returns the last weighting calculated depending on *re-cal_weighting_on_iter*.

17.8 ... Large refinements with tens of 1000s of parameters

Refinements comprising many parameters and data points can be both slow and memory intensive. Computational speed can be hindered by the **A** matrix dot products of Eq. (4-4). In the case of dense matrices, memory usage in forming the full **A** matrix can be prohibitive. The following keywords can be used to overcome these problems:

```
conserve_memory
bootstrap_errors 100
approximate_A
    A_matrix_memory_allowed_in_Mbytes 100
    A_matrix_elements_tollerance 0.00001
```

approximate_A avoids the calculation of the **A** matrix dot products; more refinement iterations are typically required for convergence but in most large problems the time to convergence is greatly decreased (see for example AE14-APPROX-A.INP). Furthermore, memory usage of the **A** matrix can be limited using *A_matrix_memory_allowed_in_Mbytes*; this produces a sparse matrix, depending on allotted memory, by removing small A_{ij} values. Typically, the calculation of the co-variance matrix is impractical and hence errors can instead be determined using the bootstrap method.

17.9 ... Laue refinement

Single crystal Laue diffraction data can be refined; data files have the extension *.HKL-LAM; see directory TEST_EXAMPLES\LAUE. *Laue_Lam* is a reserved parameter name that can be used in hkl type equations; it returns the reflection dependent wavelength. The merging of equivalent reflections and Friedel_pairs are not allowed with Laue refinement; the following keywords are internally defined with Laue refinement:

```
dont_merge_equivalent_reflections
dont_merge_Friedel_pairs
```

and the following messages reported:

```
Equivalent reflections not merged
Friedel pairs not merged
```

17.10 . Learnt Shapes for Background or Otherwise

[*xdd*]...
    [*user_y* $name { *#include* $file }]... | [*user_y* $name $file]...
        [[1]*xye_format*]
        [[1]*rebin_with_dx_of* !E]
        [[1]*user_y_hat* E]...
        [[1]*user_y_gauss_fwhm* E]...

[¹*user_y_lor_fwhm* E]...

[¹*user_y_exp_conv_const* E [*user_y_exp_limit* E]...

¹New *user_y* dependents. *user_y_hat*, *user_y_gauss_fwhm*, *user_y_lor_fwhm* and *user_y_exp_conv_const* are identical to the *hat*, *gauss_fwhm* and *lor_fwhm* and *exp_conv_const* convolutions except they are applied to the *user_y* data.

user_y can be used to add, multiply and in general manipulate data files of different x-axis steps. For example, to add two data files, square the result and then multiply by the x-axis reserved parameter *X*, the following can be used:

```
user_y f1 file1.xy
user_y f2 file2.xy
yobs_eqn result.sst = X (f1 + f2)^2; min 10 max 100 del 0.01
```

The test example USER_Y\USER_Y.INP fits five fit objects to the quartz triplet using a learnt peak shape defined using *user_y*; the fit with the individual *fit_obj*'s displayed (using the macro *Plot_Fit_Obj*) looks like:
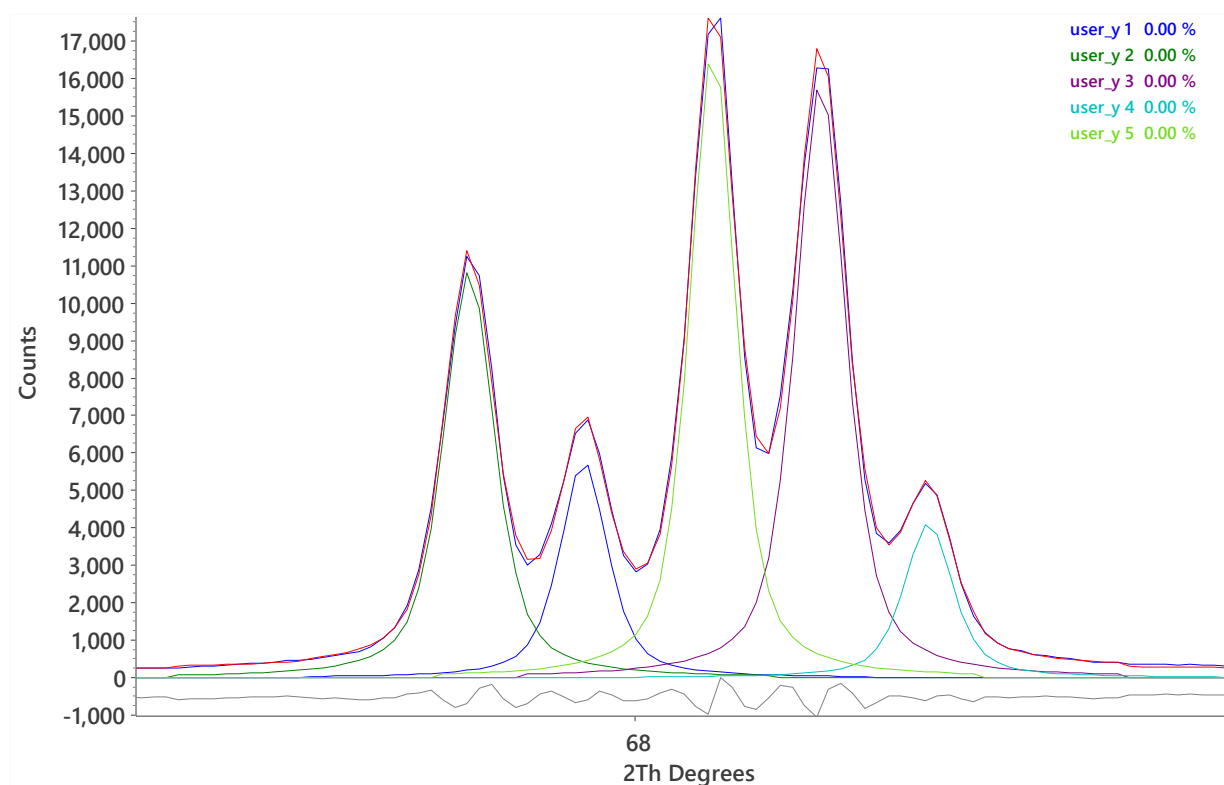
The test example USER_Y\USER_Y_CONVOLUTION.INP fits five fit objects to a simulated pattern using a learnt peak shape defined using *user_y* with the two *user_y* convolutions of *user_y_exp_conv_const* and *user_y_gauss_fwhm*; the INP file looks like:

```
'#define CREATE_SIMULATED_
continue_after_convergence

macro FO_Peak(& p, & pe, & a, & x, & s)
   {
      fit_obj = a p;
```

```
            min_X = -pe s + x; max_X = pe s + x;
                fo_transform_X = (X - x) / s;
        }

prm !peak_extent 2

#ifdef CREATE_SIMULATED_
    iters 0
    user_y peak { _xy -0.01 0 0 100 0.01 0  }
         user_y_exp_conv_const @ 1    min 0.5 max 2
         user_y_gauss_fwhm     @ 0.1  min 0.1 max 2
    yobs_eqn = 1; min 66 max 70 del 0.01
         gui_ignore ' don't load data file into GUI
         Out_X_Ycalc(user_y_convolution.xy)
#else
    ' Fit to the simulated peak
    user_y peak { _xy -0.01 0 0 100 0.01 0  }
         user_y_exp_conv_const @ 1    min 0.5 max 2 val_on_continue = Rand(0.5, 2);
         user_y_gauss_fwhm     @ 0.1 min 0.1 max 1 val_on_continue = Rand(0.1, 1);
    xdd user_y_convolution.xy
#endif
        start_X  66
        finish_X 70
        bkg @ 100
        prm a1 1000 min 1.0e-6 val_on_continue = Rand(1, 100);
        prm a2 2000 min 1.0e-6 val_on_continue = Rand(1, 100);
        prm a3 3000 min 1.0e-6 val_on_continue = Rand(1, 100);
        prm a4 2000 min 1.0e-6 val_on_continue = Rand(1, 100);
        prm a5 1500 min 1.0e-6 val_on_continue = Rand(1, 100);
        prm x1 67.7 val_on_continue = Val + Rand(-0.01, 0.01) 5; min 67 max 69
        prm x2 67.9 val_on_continue = Val + Rand(-0.01, 0.01) 5; min 67 max 69
        prm x3 68.1 val_on_continue = Val + Rand(-0.01, 0.01) 5; min 67 max 69
        prm x4 68.3 val_on_continue = Val + Rand(-0.01, 0.01) 5; min 67 max 69
        prm x5 68.5 val_on_continue = Val + Rand(-0.01, 0.01) 5; min 67 max 69
        prm s1 0.7 val_on_continue = Rand(0.5, 2); min 0.5 max 2
        prm s2 0.9 val_on_continue = Rand(0.5, 2); min 0.5 max 2
        prm s3 1.1 val_on_continue = Rand(0.5, 2); min 0.5 max 2
        prm s4 1.0 val_on_continue = Rand(0.5, 2); min 0.5 max 2
        prm s5 0.8 val_on_continue = Rand(0.5, 2); min 0.5 max 2

        FO_Peak(peak, peak_extent, a1, x1, s1) Plot_Fit_Obj("user_y 1")
        FO_Peak(peak, peak_extent, a2, x2, s2) Plot_Fit_Obj("user_y 2")
        FO_Peak(peak, peak_extent, a3, x3, s3) Plot_Fit_Obj("user_y 3")
        FO_Peak(peak, peak_extent, a4, x4, s4) Plot_Fit_Obj("user_y 4")
        FO_Peak(peak, peak_extent, a5, x5, s5) Plot_Fit_Obj("user_y 5")
```

fo_transform_X is a dependent of *fit_obj* and it can be used to transform *X* used within the *fit_obj*. This is useful for cases where the *user_y* x-axis is different to the *Yobs* x-axis. The *user_y* NAME {...} usage allow shapes to be typed directly into the INP file using the *_x1_dx* tag. A triangle for example is formulated as follows:

```
    user_y NAME {
        _x1_dx -1 1 /* start and step */
        0 1 0       /* the shape data */
    }
```

Multiple *user_y*(s) can be defined and each can be used any number of times in equations that can be a function of *X*. The test example USER_Y.INP loads a single shape and stretches and scales it five different ways onto a diffraction pattern to fit the Quartz triplet. Convergence is as fast as with any other refinement.

17.11 . Emission Profile with Absorption Edges

[*modify_peak*]
    [*modify_peak_apply_before_convolutions*]
    [*modify_peak_eqn* !E] 'Can be a fn of Get(current_peak) & Get(current_peak_x)
        [*current_peak_min_x* !E]
        [*current_peak_max_x* !E]

modify_peak can be used to modify peak profiles either before convolutions or after. Functionality is realized by using the internal data objects of Get(*current_peak_x*) and Get(*current_peak*); these two objects return the x-axis wavelength being processed and the current calculated peak intensity respectively. Here are plots from AL2O3-PAM.INP and SPINNEL-PAM.INP (see directory TEST_EXAMPLES\ABSORPTION-EDGE) that has an identical absorption edge modelled for both Al2O3 and Spinnel samples:
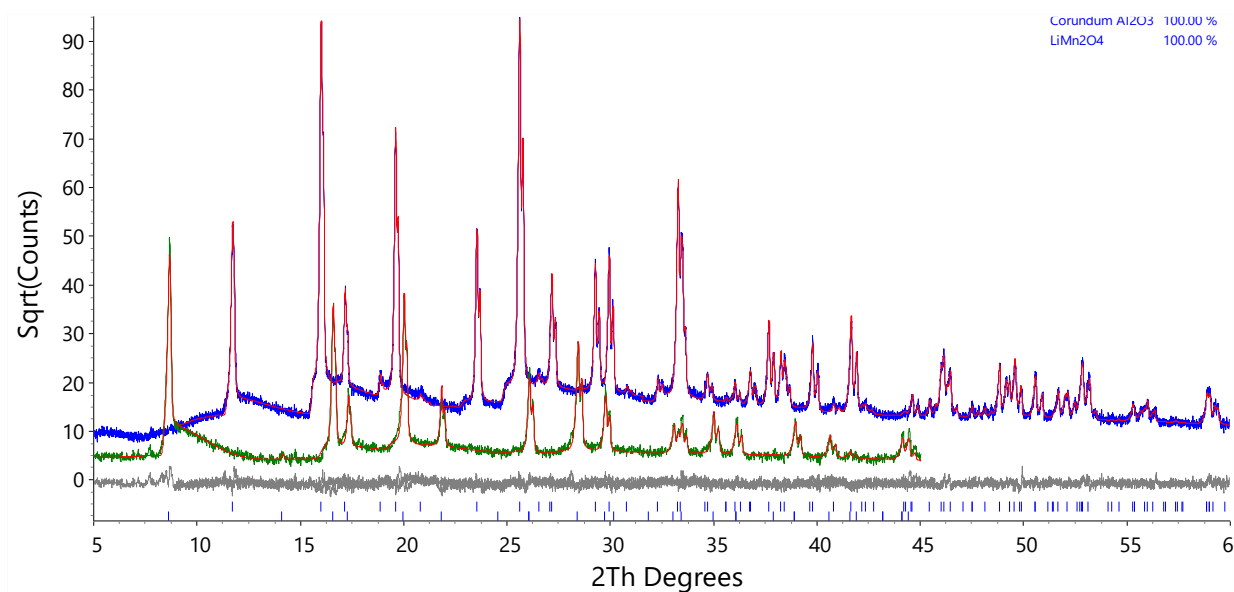
17.12 . scale_phase_X keyword

[*scale_phase_X* E]...

Scales *Ycalc* point by point. It can be used, for example, to define Lorentz Polarization factors on an x-axis basis rather than on a peak basis as is the case for *scale_pks*. Some main points for *scale_phase_X*:

- Can be a function of *X*
- Multiple definitions allowed, and each applied to the pattern.
- Can occur at the *xdd* or *phase* level.

Here's an example:

```
xdd ...
    scale_phase_X ...
    str     scale_phase_X ...
    hkl_Is  scale_phase_X ...
```

The first *str* is multiplied by the first and second *scale_phase_X*; the *hkl_ls* phase is multiplied by the first and third *scale_phase_X*.

17.13 . Refining on f0, f' and f"

[*f0_f1_f11_atom*]...
    [*f0* E][*f1* E][*f11* E]
	Examples
	
	\TEST_EXAMPLES\F0-F1-F11\
	    XRAY-POWDER.INP
	    TOF.INP

User defined atomic scattering factors, parameter *f0*, and anomalous dispersion coefficients, parameters *f1* and *f11*. Example usage:

```
report_on_str
load f0_f1_f11_atom f1 f11 {
    Ba @ -0.160127754 2.3954287
    Ge 0.184162081 1.86162161
}
```

High correlations exist between *f1*, *f11*, *scale* and *beq* parameters. *f0_f1_f11_atom* can be used at the *str*, *xdd* and global levels. *f1* and *f11* can be defined and refined independently. Defaults are used when either *f1* or *f11* are not defined. The examples XRAY-POWDER.INP and TOF.INP demonstrates the use of *f0*, *f1* and *f11*. The *f0* parameter can be a function of the reserved parameter *D_spacing*; for example:

```
prm a1 25 min -50 max 50
load f0_f1_f11_atom f0 f11 {
    Pb+2
    = a1        Exp(1.058874   (-0.25) / D_spacing^2) +
      16.496822 Exp(0.106305   (-0.25) / D_spacing^2) +
      19.984501 Exp(6.708123   (-0.25) / D_spacing^2) +
      6.813923  Exp(24.395554  (-0.25) / D_spacing^2) +
      5.233910  Exp(1.058874   (-0.25) / D_spacing^2) +
      4.065623; ' this is f0 for Pb
      @ 5        ' this is f11 for Pb
}
```

For X-ray data *f0* is by default obtained from the file ATMSCAT.CPP. For neutron data *f0* corresponds to the neutron scattering length; default scattering lengths are from the NEUTSCAT.CPP file. Neutron scattering lengths can be refined, see example TOF.INP. *no_f11* instructs the program to ignore *f11*. This increases speed with little change in *Ycalc*. *report_on_str* reports on *f1* and *f11*, or neutron scattering lengths, actually used. No values are reported when

d_spacing_to_energy_in_eV_for_f1_f11 is used. To disable the effects of *f0*, *f1* and *f11*, for say CeO2, then the following could be used:

```
load f0_f1_f11_atom f0 f1 f11 {
    Ce+4 1 0 0
    O-2  1 0 0
}
```

17.13.1......Using a user defined table to input f0 values via user_y

Atomic scattering factors *f0* can be defined in a *.XY file and used via the *user_y* keyword as follows:

```
xdd ...
    user_y C_f0_table C_f0_table.xy
    str
        load f0_f1_f11_atom f0 f1 f11 { C = C_f0_table; 0 0 }
        …
```

Here the C_F0_TABLE.XY file comprises D_spacing and f0 value pairs which is used to describe *f0* values for the C atoms withing the structure. In the above example, *f1* and *f11* are set to zero.

17.14 . Invalid f1 and f11

The following message is displayed when there are no valid entries for f' and f" in the corresponding NFF file:

```
Invalid f1 and f11 for O in file ...\ssf\o.nff
    for the wavelength 0.399826.
    Setting value(s) to zero
```

In such cases the user may choose to manually define f' and f" using *f1* and *f11* respectively. Also useful is to view f' and f" NFF files found in the ssf directory using the GUI Tools menu:
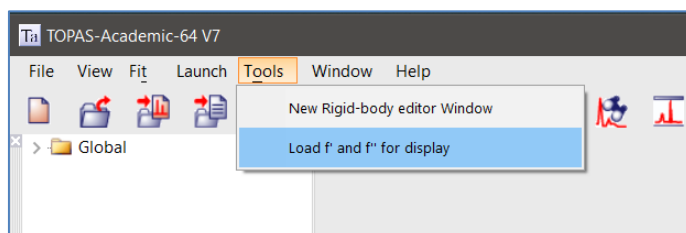
17.15 . Isotopes and Atom Names

ISOTOPES.TXT is used for obtaining isotope weights. It's possible to have the following when refining either neutron (i.e. *neutron_data* is defined) or x-ray data and to obtain the correct results without changing the INP *str*:

```
site ...  occ Mg ...
```

```
site ...  occ Mg+2 ...
site ...  occ 24Mg ...
site ...  occ 26Mg ...
site ...  occ 26Mg+2 ...
```

In the cases of 'Mg' and 'Mg+2' the atomic weight used is the 'Standard Weight' as defined in ISO-TOPES.TXT. In the cases of '26Mg' and '26Mg+2' the atomic weight used is the isotope weight as defined in ISOTOPES.TXT. Note the '+2' is dropped when searching that file. The atomic weight for 24Mg is not the same as that for Mg. When 24Mg is used then the isotope weight for 24Mg is used. When Mg is defined then the standard weight is used. The standard weight corresponds to the mean weight of the naturally occurring Mg isotopes.

In the case of x-rays:

- atomic scattering factors used (from file ATMSCAT.CPP) for 26Mg and 26Mg+2 corresponds to those of Mg or Mg+2 respectively. Numbers occurring at the start of the symbol are dropped when searching ATMSCAT.CPP.

- f' and f" corrections (files in SSF directory) corresponds to that for Mg. In other words, the numbers occurring at the start of the symbol as well as the charge (i.e. '+2' in this case) is dropped.

In the case of neutrons:

- scattering lengths used are from the NEUTSCAT.CPP file; the charge (i.e. '+2') is dropped when searching NEUTSCAT.CPP.

Internally the program converts 'D' and 'T' to '2H' and '3H' respectively.

17.16 . Atomic data files and associated sources

Table 17-2. Files read when atomic data is sought. The references refer to the source of the data. In many cases the format of the data file corresponds to the original source format.
ANOMDISP.CPP : f' and f" for Laboratory X-ray tubes. File is read if there are no associated SSF*.NFF file or if *use_tube_dispersion_coefficients* is defined.
ATMSCAT.CPP : f0 or Elastic Photon-Atom Scattering, relativistic form factors; data from http://www.esrf.fr/computing/expg/subgroups/theory/DABAX/dabax.html
ATOM_COLORS.DEF : Red, Green, Blue (RGB) CPK atom colors from http://www.bio.cmu.edu/Courses/BiochemMols/Periodic/ElemList.htm. Used for assigning colors to atoms when displaying in OpenGL.
ATOM_RADIUS.DEF : Atomic radii and Covalent radii from http://www.esrf.fr/cgi-bin/periodic.
ISOTOPES.TXT : Atomic Weights and Isotopic Compositions for All Elements from http://physics.nist.gov/PhysRefData/Compositions/

MAGDATA.DAT : Data from GSAS data file via the International tables. Data correction for V entry made by Robert Von Dreele.
NEUTSCAT.CPP : Neutron scattering lengths from http://www.ccp14.ac.uk/ccp/web-mirrors/neutrons/n-scatter/n-lengths/LIST~1.HTM
NO_POLYHEDRA.DEF : Disables drawing of polyhedral for atoms listed.
SSF*.NFF : Anomalous scattering factors f' and f" for a range of wavelengths from http://www-cxro.lbl.gov/optical_constants/asf.html The present data is in three columns "E(eV),f1,f2" where f'=f1–Z and f''= f2 and the conversion from wavelength to energy scale is: E(eV)=10^5/(8.065541*Lambda(Ang)).
MAC\\ZNN.HTML : X-Ray Mass Attenuation Coefficients from http://www.nist.gov/pml/data/xraycoef/index.cfm

17.17 . Removing Phases during a refinement

[*remove_phase* !E]

Allows for phase removal during refinement, see TEST_EXAMPLES\REMOVE-PHASE.INP which uses the *Remove_Phase* macro. Typical use is:

```
for strs {
    Remove_Phase(0.3, 0.5)
}
```

Here a phase is removed if its weight percent is below 0.3% and if the error in the weight percent is greater than 0.5 times the weight percent. The phase removal process is executed at the end of a *Cycle*. Text similar-to the following is displayed on removal of a phase:

```
*** Deleting phase: Corundum ***
*** Deleting phase: Zincite ***
    etc...
```

Refinement is terminated when no phases are removed during a *Cycle*.

17.18 . Numerical Lorentzian and Gaussian Convolutions

For fundamental and pseudo-Voigt peak types, Lorentzian and Gaussian convolutions are performed analytically during the calculation of the emission profile Voigt. Therefore, *lor_fwhm* and *gauss_fwhm* are still calculated at the emission profile level even when defined between *push_peak* and *add_pop_1st_2nd_peak* keywords.

17.19 . Space groups, hkls and symmetry operators

[*space_group* $symbol]

Used to define the space group where $symbol can be any space group symbol (case insensitive) occurring in the file SGCOM5.CPP, it can also be a space group number; here are some examples:

```
space_group "I a -3"
space_group ia-3
space_group P_63_M_C
space_group I_41/A_M_D
space_group I_41/A_M_D:2 ' defines second setting of I_41/A_M_D
space_group 206
space_group 222:2        ' defines second setting of 222
```

Symmetry operators are generated by SGCOM6.EXE and placed into a SG*.SG file with a name similar-to the name of the space group. Space group details for space groups with names containing the characters '/' or ':' are placed in files with file names similar-to the space group but with the characters replaced by 'o' and 'q' respectively. The reason for this is that file names containing these characters are not allowed on some operating systems. hkl generation uses information in the *.sg file.

17.19.1......User defined rotational matrices

Space group generator - User defined rotational matrices can be added to the file SGROTS3.CPP found in the main TA directory.

17.20 . Site identifying strings

Keywords such as *operate_on_points* use a site identifying string which can contain the wild card character '*' and a negation character '!'. The wild card character '*' used in "O*" means that sites with names starting with 'O' are considered. In addition to using the wild card character, site names can be written explicitly within double quotation marks. Table 17-3 shows some *operate_on_points* strings and the corresponding sites identified.

Table 17-3. Example *operate_on_points* strings and corresponding sites identified.

str	$sites	Sites identified
site Pb1 ...	*	Pb1, S1, O1, O2, O31, O32, O4
site S1 ...	Pb*	Pb1
site O1 ...		
site O2 ...	"Pb1 S*"	Pb1, S1
site O31 ...	O*	O1, O2, O31, O32, O4
site O32 ...		
site O4 ...	"O* !O3*"	O1, O2, O4
	"O* !O1 !O2"	O31, O32, O4

17.21 . Occupancies and symmetry operators

Only unique positions are generated from symmetry operators. Fully occupied sites therefore require site occupancy values of 1. A comparison of atomic positions is performed in the generation of the unique positions with a tolerance in fractional coordinates of 10^{-15}. It is therefore necessary to enter fractions in the form of equations when entering fractional atomic coordinates that have recurring values such as 0.33333..., 0.66666... etc., for example:

use:	x = 1/3; y = 1/3; z = 2/3;
instead of:	x 0.33333 y 0.33333 z 0.66666

17.22 . Pawley and Le Bail extraction

[*lebail* #]

For Le Bail intensity extraction (see example LEBAIL1.INP) use the input segment:

```
    hkl_Is    space_group p-1  lebail 1 ...
```

For Pawley intensity extraction (see example PAWLEY1.INP) use the input segment:

```
    hkl_Is    space_group p-1 ...
```

hkls are generated in the absence of *hkl_m_d_th2* keywords. After refinement, the details for the generated hkl's are appended after *space_group*. For the Pawley method, once the hkl details are generated, parameter equations can be applied to the *I* parameters as usual.

17.23 . Anisotropic refinement models

Keywords that can be a function of *H*, *K*, *L* and *M*, as shown in Table 2-3, allow for the refinement of anisotropic models including preferred orientation, and peak broadening. An important consideration when dealing with hkls in equations is whether to work with hkls or whether to work with their multiplicities. The *Multiplicities_Sum* macro can be used when working with multiplicities, for example:

```
    prm a 0 th2_offset = Multiplicities_Sum(If(Mod(L, 2) == 0, a Tan(Th), 0));
```

L here corresponds to the *L*'s of the multiplicities. Note, the preferred orientation macro *PO* uses the *Multiplicities_Sum* macro and *Spherical Harmonics* uses the hkls in the *.hkl file only. A completely different viewpoint than to refine on half widths is to consider the distribution of lattice metric parameters within a sample. Each crystallite is regarded as having its own lattice parameters, with a multi-dimensional distribution throughout the powder sample. This can be achieved by adding the same structure several times to the input file.

17.23.1......Spherical harmonics

spherical_harmonics_hkl can be applied to both peak shapes, for anisotropy, and intensities for a preferred orientation correction. Preferred orientation can be described using the *PO_Spherical_Harmonics*(sh, order) macro, where "sh" is the parameter name and "order" the order of the spherical harmonics. *scale_pks* is used to correct peak intensities as follows:

```
     macro PO_Spherical_Harmonics(sh, order) {
        spherical_harmonics_hkl sh
           sh_order order
           scale_pks = sh;
     }
```

Example clay.inp uses *spherical_harmonics_hkl* for describing anisotropic peak broadening using the *exp_conv_const* convolution as follows:

```
     str ...
        spherical_harmonics_hkl sh
           sh_order 8
        exp_conv_const = (sh-1) Tan(Th);
```

17.23.2......Miscellaneous models using User defined equations

Anisotropic Gaussian broadening as a function of L (see example CEO2HKL.INP):

```
     str ...
        prm a 0.1 min 0.0001 max 5
        prm b 0.1 min 0.0001 max 5
        gauss_fwhm = If(L==0, a Tan(Th) + 0.2, b Tan(Th));
```

Anisotropic peak shifts as a function of L (*th2_offset*):

```
     str ...
        prm at 0.07 min 0.0001 max 1
        prm bt 0.07 min 0.0001 max 1
        th2_offset = If(L == 0, at Tan(Th), bt Tan(Th));
```

Description of anisotropic peak broadening using the March (1932) relation and *str_hkl_angle*:

```
     str ...
        str_hkl_angle ang1 1 0 0
        prm p1 1    min 0.0001 max 2
        prm p2 0.01 min 0.0001 max 0.1
        lor_fwhm = p2 Tan(Th) Multiplicities_Sum(((p1^2 Cos(ang1)^2 +
                   Sin(ang1)^2 / p1)^(-1.5)));
```

17.24 . Simulated annealing and structure determination

Defining *continue_after_convergence* and a temperature regime is analogous to defining a simulated annealing process (Coelho, 2000). After convergence, a new refinement cycle is initiated with parameter values changed according to any defined *val_on_continue* attributes and *rand_xyz* or *randomize_on_errors* processes. Simulated annealing is therefore not specific to structure solution, see for example ONLYPENA.INP and ROSENBROCK-10.INP. Convergence is determined when the change in χ^2 is less than *chi2_convergence_criteria* for three consecutive cycles and when all defined *stop_when* parameter attributes evaluate to true. Example:

```
     chi2_convergence_criteria = If(Cycle_Iter < 10, 0.001, 0.01);
```

For structure solution in real space, the need for computation efficiency is critical. In many cases computation speed can be increased by up to a factor of 20 or more with the appropriate choice of keywords. Keywords that facilitate speed are:

```
chi2_convergence_criteria...
quick_refine...
yobs_to_xo_posn_yobs...
```

Another category is one that facilitate structure solution by changing the form of χ^2:

```
penalties_weighting_K1...
penalty...
occ_merge...
rigid...
```

Further keywords and processes typically used are:

```
file_name_for_best_solutions
seed
temperature !E ...
   move_to_the_next_temperature_regardless_of_the_change_in_rwp
   save_values_as_best_after_randomization
   use_best_values
xdd ... or xdd_scr ...
   str ...
      site ... rand_xyz ...
```

17.24.1......Penalties used in structure determination

Introducing suitable penalties can reduce the number of local minima in χ^2 and correspondingly increase the chances of obtaining a global minimum. The structure factor for a reflection with Miller indices 10 0 0 for a two-atom triclinic unit cell with fractional atomic coordinates of (0,0,0) and (x, 0,0) is given by $4\cos(\pi hx)^2$; here there are 10 local minima for 0<x<1. If it was known that the bond length distance is half the distance of the *a* lattice parameter then a suitable penalty would reduce the number of minima to one. In this trivial example the number of minima increases as the Miller indices increase. For non-trivial structures and for the important d spacing range near inter-atomic distances of 1 to 2Å the number of local minima is very large. Bragg reflections with large Miller indices that are heavily weighted are expected to contain many false minima; by applying an appropriate weighting scheme to the diffraction data the search for the global minimum can be facilitated. For powder data the default *weighting* scheme is:

```
weighting = If(Yobs <= 1, 1, 1 / Yobs);
```

For single crystal data the following, which is proportional to 1/d, works well:

```
weighting = 1 / (Sin(X Deg / 2) Max(Yobs,1));
```

A more elaborate scheme which also works well for single crystal data is:

```
weighting = ( Abs(Yobs-Ycalc) / Abs(Yobs+Ycalc) + 1) / Sin(X Deg / 2);
```

Two penalty functions that have shown to facilitate the determination of structures are the anti-bumping (AB) penalty and the potential energy penalty U. The anti-bumping penalty is written as:

$$AB_i = \begin{cases} \sum (r_{ij} - r_o)^2, & \text{for } r_{ij} < r_o \text{ and } i \neq j \\ 0, & \text{for } r_{ij} > r_o \end{cases} \qquad \text{(17-14)}$$

where r_0 is a bond length distance, r_{ij} the distance between atoms i and j including symmetry equivalent positions and the summation is over all atoms of type j. The *ai_anti_bump* and *box_interaction* keywords are used to implement the penalty of Eq. (17-15) using the *AI_Anti_Bump* and *Anti_Bump* macros respectively. Typically, Anti bump constraints are applied to heavy atoms; an over use of such constraints can in fact hinder simulated annealing in finding the global minimum. Applying the constraint for the first few iterations of a refinement cycle can also be beneficial; this is achieved in the *AI_Anti_Bump* macro by writing the penalty in terms of the reserved parameter *Cycle_Iter*; see for example CIME-DECOMPOSE.INP.

The *grs_interaction* can be used to calculate the Lennard–Jones or Born–Mayer potentials and it is suited to ionic atomic models (see example ALVO4-GRS-AUTO.INP). For a site i they comprise a Coulomb term C_i and a repulsive term *Ri* and is written as:

$$U_i = C_i + R_i \qquad \text{(17-15)}$$

$$\text{where} \quad C_i = A \sum_{i,j} \frac{Q_i Q_j}{r_{i,j}}, \quad i \neq j$$

$$R_i = \sum_{i,j} \frac{B_{i,j}}{r_{i,j}^n}, \quad \text{for Leonard Jones and } i \neq j$$

$$R_i = \sum_{i,j} c_{i,j} \exp(-d\, r_{i,j}), \quad \text{for Born} - \text{Mayer and } i \neq j$$

where $A = e^2/(4\pi\varepsilon_0)$ and $\varepsilon 0$ is the permittivity of free space, Q_i and Q_j are the ionic valences of atoms i and j, $r_{i,j}$ is the distance between atoms i and j and the summation is over all atoms to infinity. The repulsive constants $B_{i,j}$, n, $c_{i,j}$ and d are characteristic of the atomic species and their potential surrounds. The equation part of the *grs_interaction* is typically used to describe the repulsive terms.

17.24.2......Bond length restraints

Example ALVO4-GRS-AUTO.INP defines a bondlength restraint using the GRS series between an Aluminum site and three Oxygen sites. Valence charges have been set to +3 and –2 for Aluminum and Oxygen, respectively. The expected bond length is 2 Å between for O-O bonds and 1.5 Å for Al-O bonds:

```
    site Al  x @ 0.7491  y @ 0.6981  z @ 0.4069  occ Al+3 1  beq 0.25
    site O1  x @ 0.6350  y @ 0.4873  z @ 0.2544  occ  O-2 1  beq 1
    site O2  x @ 0.2574  y @ 0.4325  z @ 0.4313  occ  O-2 1  beq 1
    site O3  x @ 0.0450  y @ 0.6935  z @ 0.4271  occ  O-2 1  beq 1
    Grs_Interaction(O*, O*, -2, -2, oo,  2.0, 5)  penalty = oo;
    Grs_Interaction(Al, O*,  4, -2, alo, 1.5, 5)  penalty = alo;
```

The following example defines a bondlength restraint using the *Al_Anti_Bump* macro between a Potassium site and three Carbon sites. The expected bond length is 4 Å between Potassium sites and 1.3 Å between Carbon sites.

```
    site K   x @ 0.14305  y @ 0.21812  z @ 0.12167  occ K 1  beq 1
    site C1  x @ 0.19191  y @ 0.40979  z @ 0.34583  occ C 1  beq 1
    site C2  x @ 0.31926  y @ 0.35428  z @ 0.32606  occ C 1  beq 1
    site C3  x @ 0.10935  y @ 0.30991  z @ 0.39733  occ C 1  beq 1
    AI_Anti_Bump(K , K , 4  , 1)
    AI_Anti_Bump(C*, C*, 1.3, 1)
```

Unlike the first example, there's no explicit definition of a penalty function as the *Al_Anti_Bump* macro includes the penalty function.

17.25 . File types and formats

Table 17-4. File types.	
*.PRO	Project files.
*.INP	Input file in INP format.
*.OUT	Output file created on termination of refinement in INP format.
*.STR	Structure data. Same format as *.INP.
*.LAM	Source emission profile data. Same format as *.INP.
*.DEF	Program defaults. Same format as *.INP.
*.LOG	TOPAS.LOG and TC.LOG. Useful for tracking input errors.
Measurement Data	
*.SST	Implies an equal x-axis and has the folformat of *"start, step, data points...."* SST files can be used instead of *.XY files. As x-axis values are not used, they save space on creation as well as on loading. For equal x-axis data then the macro *Out_XDD_SST* can be used in the following manner: ```xdd …\n Out_XDD_SST(filename.sst) = Ycalc; ' outut Ycalc\n Out_XDD_SST(filename.sst) = Yobs − Ycalc; ' output difference plot```
*.RAW	Bruker AXS binaries (DIFFRAC AT and DIFFRAC*plus*)
*.DAT, *.XDD, *.CAL, *.XY, *XYE	ASCII file formats, see Table 17–5
*.SCR	ASCII file format comprising lines of h, k, l, m, d, 2θ, and Fo.
*.HKL	ShelX HKL4 format.
Structure and structure factor data	

*.CIF	Crystallographic Information File; International Union for Crystallography.
*.FCF	CIF file representation of structure factor details suitable for generating Fourier maps using ShelX.

Table 17-5. ASCII input data file formats. *.XY, *.XYE, *.XDD and *.CAL are white space delimited and can contain line and block comments.

*.DAT, LHPM/RIET7/CSRIET		
	Line 1–4	Comments
	Line 5	Start, Step and Finish angle
	Line 6 ...	Observed XRD data points
GSAS ("std - const", "alt - ralf"), use *gsas_format*		
	Line 1	Legend
	Line 2	Item 3: Number of data points
	Line 3 ...	Depending on item10 and item5

	For item10 = "STD" and item5 = "CONST"
	    xmin = item6/div
	    step =item7/div
	    read(10(i2,F6.0) iww(i),y(i) i=1, npts
	    sigma(i)=sqr(y(i)/iww(i)) i=1, npts
	For item10 = "ALT" and item5 = "RALF"
	    xmin = item6/32
	    step = item7/32
	    read(4(F8.0,F7.4,F5.4) x(i), y(i), sigma(i) i=1, npts
	    x(i) = x(i)/32 i=1, npts
	    do i = 1, npts-1
	      div = x(i+1)–x(i)
	      y(i) =1000 * y(i)/div
	      sigma(i) = 1000 * sigma(i)/div
	    end do
	rk (constant wavelength data): div = 100
	rk (time of flight data): div = 1	

FullProf (INSTRM = 0: free format file), use *fullprof_format*		
	Line 1	Start angle, step width, finish angle, comments
	Line 2 ...	Observed XRD data points (any number of rows)
*.XDD, *.CAL	Line 1	Optional line for comments
	Line 2 ...	Start, Step and Finish angle
		Next three numbers are unused
		Observed XRD data points
*.XY		2θ and intensity data values
*.XYE		2θ, intensity and intensity error values.

17.26 . Batch mode operation – TC.EXE

The command line program tc.exe provides for batch mode operation. Running tc.exe without arguments displays help information. Running an INP file called PBSO4.INP is as follows:

```
tc pbso4
```

Macros can be passed to the command line. Passing a file name to an INP file is as follows:

1) Create a TEMPLATE.INP file with the required refinement details, this could look like the following:

```
xdd FILE
    etc...
```

2) TEMPLATE.INP is fed to TC.EXE at the command line; the word FILE (within TEMPLATE.INP) is expanded to whatever the macro on the command line comtains. For example:

```
tc ...\file_directory\template.inp "macro FILE { file.xy }"
```

The macro called FILE is described on the command line within quotation marks. On running tc.exe the word 'FILE' occurring in TEMPATE.INP is expanded to 'file.xy'. Note that more than one macro can be described on the command line. To process a whole directory of data files, say *.XY file for example, then:

1) Execute the following DOS command from the file directory:

```
dir *.xy > ...\main_ta_directory\xy.bat
```

The XY.BAT file will then reside in the main TA directory.

2) Edit ...\MAIN_TA_DIRECTORY\XY.BAT to look like the following:

```
tc ...\file_directory\template "macro FILE { file1.xy }"
copy ...\file_directory\template.out ...\file_directory\file1.out
tc ...\file_directory\template.inp "macro FILE { file2.xy }"
copy ...\file_directory\template.out ...\file_directory\file2.out
etc...
```

After each run of TC.EXE a template.out file is created containing refined results. This file is copied to another file FILE1.OUT, FILE2.OUT etc... to avoid it from being overwritten. After running XY.BAT a number of *.OUT files is created one for each *.XY file. In summary tc.exe receives TEMPLATE.INP to process. Words occurring in TEMPLATE.INP are expanded depending on the macros described on the command line.

18. KEYWORDS

18.1 ... Data structures

The following describes keyword dependencies. Trailing '...' implies that more than one node of that type can be inserted under its parent. Items enclosed in square brackets are optional. Items beginning with a capital T corresponds to keyword groups analogous to complex types in XML.

Ttop
    Tcomm_1
    Tcomm_2
    Ttop_xdd
    Tglobal
    Txdd
    Txdd_scr
    Tindexing
    Tcharge_flipping

Ttop_xdd
    [*convolution_step* #1]
    [*Rp* !E][*Rs* !E]
    [*x_calculation_step* !E]

Tglobal
    TMinimization
    Trwp
    [*A_matrix*][*C_matrix*][*A_matrix_normalized*][*C_matrix_normalized*]
    [*conserve_memory*]
    [*file_name_for_best_solutions* $file]
    [*force_positive_fwhm*]
    [*inp_text* $name]...[*inp_text_insert* $name { ... }]...
    [*iters* #]
    [*no_LIMIT_warnings*]
    [*num_cycles* #]
    [*out_A_matrix* $file]
    [*out_refinement_stats*]
    [*out_rwp* $file]
    [*out_prm_vals_per_iteration* $file]...|[*out_prm_vals_on_convergence* $file]...
    [*process_times*]
    [*randomise_file_out_normal* $file]
    [*seed*[#]]
    [*suspend_writing_to_log_file* #1]
    [*temperature* !E]...
    [*use_tube_dispersion_coefficients*]
    [*verbose* #1]

Txdd
    [*xdd* $file [{$data}] [*range* #] [*xye_format*] [*gsas_format*] [*fullprof_format*]]...
        Ttop_xdd
        Txdd_comm_1
        Tcomm_1
        Tcomm_2
        Tmin_max_rc
        Trwp
        [*gui_add_bkg* !E]
        [*xdd_sum* !E] and [*xdd_array* !E]
        [*xo_ls*]...
            [*xo* E *I* E]...
            Tcomm_1_2_phase_1_2
        [*d_ls*]...
            [*d* E *I* E]...
            Tcomm_1_2_phase_1_2
            [*lebail* #]
        [*hkl_ls*]...
            [*lp_search* !E]
            [*I_parameter_names_have_hkl* $start_of_parameter_name]
            [*hkl_m_d_th2* # # # # # # E *I* E]...
            Tspace_group
            Tcomm_1_2_phase_1_2
            Thkl_lat
            [*lebail* #]
        [*str* | *dummy_str*]...
            Tstr_details
            Thkl_lat
            Tcomm_1_2_phase_1_2
            Tmin_max_rs
            [*rigid*]...
            Tspace_group

Tcomm_1_2_phase_1_2
    Tcomm_1
    Tcomm_2
    Tphase_1
    Tphase_2

Txdd_scr
    [*xdd_scr* $file]...
        Txdd_comm_1
        Tcomm_2
        Ttop_xdd
        Tmin_max_r
        [*str*]...
            Tstr_details

Tphase_1
Tcomm_2
Thkl_lat
Tmin_max_r
[*rigid*]...
Tspace_group
Tscr_1

Tscr_1
[Flack E]
[*i_on_error_ratio_tolerance* #]
[*num_highest_I_values_to_keep* #]

Txdd_comm_1
[*bkg* [@] # # #...]
[*degree_of_crystallinity* #]
[*d_spacing_to_energy_in_eV_for_f1_f11* !E]
[*exclude* #ex1 #ex2]...
[*extra_X_left* !E] [*extra_X_right* !E]
[*fit_obj* E [*min_X* !E] [*max_X* !E]]...
[*neutron_data*]
[*rebin_with_dx_of* !E]
[*smooth* #]
[*start_X* !E] [*finish_X* !E]
[*weighting* !E [*recal_weighting_on_iter*]]
[*xdd_out* $file [*append*]]...
    Tout_record
[*yobs_eqn* !N E]
[*yobs_to_xo_posn_yobs* !E]

Tcomm_1
[*axial_conv*]...
[*capillary_diameter_mm* E]...
[*lpsd_th2_angular_range_degrees* E]...
[*circles_conv* E]...
[*exp_conv_const* E [*exp_limit* E]]...
[*ft_conv* E]...
[*gauss_fwhm* E]...
[*h1* E *h2* E *m1* E *m2* E]
[*hat* E [*num_hats* #1]]...
[*modify_peak*]
[*more_accurate_Voigt*]
[*lor_fwhm* E]...
[*numerical_lor_gauss_conv*]
[*numerical_lor_ymin_on_ymax* #0.0001]
[*one_on_x_conv* E]...
[*pk_xo* E]
[*push_peak*]...

[*pv_lor* E *pv_fwhm* E]
[*spv_h1* E *spv_h2* E *spv_l1* E *spv_l2* E]
[*stacked_hats_conv* [*whole_hat* E [*hat_height* E]]...[*half_hat* E [*hat_height* E]]...]...
[*th2_offset* E]...
[*user_defined_convolution* E *min* E *max* E]...
[*WPPM_ft_conv* E]...

Tcomm_2
[*f0_f1_f11_atom*]...
[*lam* [*ymin_on_ymax* #][*no_th_dependence*][*Lam* !E][*calculate_Lam*]]
[*scale_pks* E]...
[*scale_phase_X* E]
[*prm*|*local* E [*min* !E][*max* !E][*del* !E][*update* !E][*stop_when* !E][*val_on_continue* !E]]...
[*existing_prm* E]...
[*penalty* !E]...
[*out* $file [*append*]]...
        Tout_record

Tphase_1
[*atom_out* $file [*append*]]...
        Tout_record
[*auto_scale* !E]
[*del_approx* !E]
[*phase_name* $phase_name]
[*phase_out* $file [*append*]]...
[*phase_out_X* $file [*append*]]...
[*brindley_spherical_r_cm* !E]
[*r_bragg* #]
[*remove_phase* !E]
[*scale* E]

Tphase_2
[*peak_buffer_step* E [*report_on*]]
[*peak_type* $type]
[*numerical_area* E]

Tstr_details
[*append_cartesian*][*append_fractional* [*in_str_format*]]
[*append_bond_lengths* [*consider_lattice_parameters*]]
[*atomic_interaction* N E]|[*ai_anti_bump* N]...
[*box_interaction* [*from_N* #][*to_N* #][*no_self_interaction*] $site_1 $site_2 N E]...
[*fourier_map* !E]
[*grs_interaction* [*from_N* #][*to_N* #][*no_self_interaction*] $site_1 $site_2 *qi* # *qj* # N E]...
[*hkl_plane* $hkl]...
[*no_f11*]
[*normalize_FCs*]
[*occ_merge* $sites [*occ_merge_radius* !E]]...
[*p1_fractional_to_file* $file][*in_str_format*]...

[*site* $site]...
    [*adps*][*u11* E][*u22* E][*u33* E][*u12* E][*u13* E][*u23* E]
    Tmin_r_max_r
[*sites_distance* N]|[*sites_angle* N]|[*sites_flatten* N [*sites_flatten_tol* !E]]...
[*sites_geometry* N]...
[*siv_s1_s2* # #]
[*report_on_str*]
[*view_structure*]

Thkl_lat
    [*a* E][*b* E][*c* E][*al* E][*be* E][*ga* E]
    [*normals_plot* !E]...
    [*phase_penalties* $sites N [*hkl_Re_Im* #h #k #I #Re #Im]...]...
    [*spherical_harmonics_hkl* $name]...
    [*str_hkl_angle* N h k I]...
    [*omit_hkls* !E]

Tout_record
    [*out_record*]...

Tmin_r_max_r
    [*min_r* #][*max_r* #]

Tspace_group
    [*space_group* $symbol]

Miscellanous
    [*aberration_range_change_allowed* !E]
    [*default_I_attributes* !E]
    load, *move_to*, *for*

18.2 ... Alphabetical listing of keywords

[*a* E][*b* E][*c* E][*al* E][*be* E][*ga* E]

Lattice parameters in Å and lattice angles in degrees.

[*adps*][*u11* E][*u22* E][*u33* E][*u12* E][*u13* E][*u23* E]

Use of *adps* generates the *unn* atomic displacement parameters with considerations made for special positions. On termination of refinement the *adps* keyword is replaced with the *unn* parameters; see example AE1-ADPS.INP. Instead of using the *adps* keyword the *unn* parameters can be manually entered. The *unn* matrix can be kept positive definite with the site dependent macro *ADPs_Keep_PD*; this can stabilize refinement. The *ADPs_Keep_PD* macro can be used after the *unn* parameters are created. For determining adp constraints the 3x3 eigen value determination routine of Kopp (2006) has been used.

amorphous_phase[*A_matrix*][*C_matrix*][*A_matrix_normalized*][*C_matrix_normalized*]

Generates the un-normalized and normalized A and correlation matrices. If *do_errors* is defined then *C_matrix_normalized* is automatically generated and appeneded to the OUT file.

[*append_cartesian*][*append_fractional*[*in_str_format*]]

Appends site fractional coordinates in Cartesian or fractional coordinates respectively to the *.OUT file at the end of a refinement. For the case of *append_fractional*, *in_str_format* formats the output in INP format.

[*append_bond_lengths*[*consider_lattice_parameters*]]

Appends bond lengths to the end of the *.OUT file at the end of refinement. A number corresponding to equivalent positions is appended to site names. *consider_lattice_parameters* includes lattice parameter errors in the calculation of bond length and bond angle errors. An example of bond lengths output is as follows:

```
Y1:0     O1:0     2.23143
         O2:0     2.23143     88.083
         O3:0     2.28045     109.799     99.928
```

The first line gives the distance between the sites Y1 and O2. The first number in the second line gives the distance between sites Y1 and O2. The third number of 88.083 gives the angle between the vectors Y1 to O1 and Y1 to O2. The first number on the third line contains the distance between sites Y1 and O3. The second number in the third line contains the angle between the vectors Y1 to O3 and Y1 to O2. The third number in line three contains the angle between the vectors Y1 to O3 and Y1 to O1. Bond lengths, therefore, correspond to the first number in each line and bond angles start from the second number. The numbers after the site name and after the ':' character corresponds to the site equivalent position as found in the *.SG space group files found in the SG directory

[*atomic_interaction* N E]| [*ai_anti_bump* N]...
    ai_sites_1 $sites_1 *ai_sites_2* $sites_2
    [*ai_no_self_interation*]
    [*ai_closest_N* !E]
    [*ai_radius* !E]
    [*ai_exclude_eq_0*]
    [*ai_only_eq_0*]

Defines an atomic interaction with the name N between sites identified by $site_1 and $site_2. For *atomic_interaction*, E is the site interaction equation that can be a function of R and Ri. R returns the distance in Å between two atoms; these distances are updated when dependent fractional atomic coordinates are modified. The name of the *atomic_interaction* N can be used in equations including penalty equations. For *ai_anti_bump,* an internal c++ anti-bump interaction equation is used. For anti-bumping only, *ai_anti_bump* is faster than using *atomic_interaction*. The macro *AI_Anti_Bump* uses *ai_anti_bump*. *no_self_interaction* prevents interactions between equivalent positions of a site. This is useful when a general position is used to describe a special position.

ai_closest_N: interactions between $sites_1 and $sites_2 are sorted by distance and only the first *ai_closest_N* number of interactions are considered.

ai_radius: only the interactions between $sites_1 and $sites_2 that are within the distance *ai_radius* is considered.

When *ai_radius* and *ai_closest_N* are both defined then interactions from both sets of corresponding interaction are considered.

ai_exclude_eq_0: only interactions that is not the first equivalent positions in $sites_2 are considered. For example, in the following:

```
atomic_interaction ...
    ai_exclude_eq_0
    ai_sites_1 Pb
    ai_sites_2 "O1 O2"
```

the following interactions are considered:

```
Pb:0 and O1:n      (n ≠ 0)
Pb:0 and O2:n      (n ≠ 0)
```

where the number after the ':' character corresponds to the equivalent positions of the sites. *ai_only_eq_0*: only interactions between equivalent positions 0 are considered.

Functions

The *atomic_interaction* equation can be a function of the following functions:

AI_R(#ri): Returns the distance between the current site and the atom defined with *Ri*=#ri.

AI_R_CM: A function of no arguments that returns the geometric center of the current atom and the atoms defined in $sites_2.

AI_Flatten(#toll): A function that returns the sum of distances of the current atom and those defined in $sites_2 to an approximate plane of best fit. The plane of best fit is constructed such that the sum of the perpendicular distances to the current atom plus those defined in $sites_2 are a minimum

AI_Cos_Angle(#ri1, #ri2): Returns the Cos of the angle between the atom define as *Ri*=#ri1, the current atom and the atom defined as *Ri*=#ri2.

AI_Angle(#ri1, #ri2) : Similar-to AI_Cos_Angle except that the value returned is the angle in degrees.

Examples BENZENE_AI1.INP, BENZENE_AI2.INP and BENZENE_AI3.INP demonstrates the use of the *atomic_interaction* functions. *atomic_interaction*'s are used to apply geometric restraints. For example, anti-bumping between molecules for the first ten iterations of a refinement cycle can be formulated as follows:

```
        atomic_interaction ai1 = If(R < 3, (R-3)^2, 0);
            ai_exclude_eq_0
            ai_sites_1 C*
            ai_sites_2 C*
            ai_radius 3
        penalty = If(Cycle_Iter < 10, ai1, 0);
```

[*atom_out* $file [*append*]]...

Used for writing *site* dependent details to file. See *out* for a description of *out_record*. The *Out_CIF_STR* macro uses *atom_out*

[*axial_conv*]...
    filament_length E *sample_length* E *receiving_slit_length* E
    [*primary_soller_angle* E]
    [*secondary_soller_angle* E]
    [*axial_n_beta* !E]

Full axial divergence model using the method of Cheary & Coelho (1998b). *filament_length*, *sample_length* and *receiving_slit_length* define the lengths of the tube-filament, sample and receiving slit in the axial plane in mm. *primary_soller_angle* and *secondary_soller_angle* define Soller slit angles in degrees. *axial_n_beta* defines the number of rays emanating from a point X-ray source in the axial plane. Larger values for *axial_n_beta* increases both accuracy and calculation time. The macro *Full_Axial_Model* simplifies the use of *axial_conv*.

[*bkg* [@] # # # ...]

Defines a Chebyshev polynomial where the number of coefficients is equal to the number of numeric values appearing after *bkg*.

[*box_interaction* [*from_N* #] [*to_N* #] [*no_self_interaction*] $site_1 $site_2 N E]...

Defines a site interaction with the name N between sites identified by $site_1 and $site_2. E represents the site interaction equation which can be a function of *R* and *Ri*. *R* returns the distance in Å between two atoms; these distances are updated when dependent fractional atomic coordinates are modified. The name of the *box_interaction* N can be used in equations including penalty equations. When either *from_N* or *to_N* are defined, the interactions between $site_1 and $site_2 are sorted by distance and only the interactions between the *from_N* and *to_N* are considered. *no_self_interaction* prevents any interactions between equivalent positions of the same site. This is useful when a general position is used to describe a special position. For example, the following could be used to iterate from the nearest atom to the third atom from a site called Si1:

```
    str
        site Si1 ...
        site O1 ...
        site O2 ...
        site O3 ...
        box_interaction Si1 O* to_N 2 !si1o = (R-2)^2;
        penalty = !si1o;
```

In this example the nearest three oxygen atoms are soft constrained to a distance of 2 Å by the use of the penalty function. Counting starts at zero and thus *to_N* is set to 2 to iterate up to the third nearest atom.

The wild card character '*' used in "O*" means that sites with names starting with 'O' are considered. In addition to using the wild card character, the site names can be explicitly written within double quotation marks, for example:

```
box_interaction Si1 "O1 O2 O3" to_N 3 etc...
```

Interactions between Si1 and the three oxygen atoms O1, O2, O3 may not all be included, for example, if Si1 had as its nearest neighbours the following:

Si1-O1,1 at a distance of 1.0 Å

Si1-O2,3 at a distance of 1.1 Å

Si1-O2,1 at a distance of 1.2 Å

Si1-O1,2 at a distance of 1.3 Å

then two equivalent positions of site O1 and two equivalent positions of O2 are included in the interaction equation; thus, interactions between Si1-O3 are not considered. To ensure that each of the three oxygens had Si1 included in an interaction equation then the following could be used:

```
box_interaction "O1 O2 O3" Si1 to_N 0 etc...
```

Thus, the order of $site_1 and $site_2 is important when either *from_N* or *to_N* is defined. The reserved parameters *Ri* and *Break* can also be used in interaction equations when either *from_N* or *to_N* is defined. *Ri* returns the index of the current interaction being operated on with the first interaction starting at *Ri* = 0.

box_interaction is used for example by the *Anti_Bump* macro.

[*brindley_spherical_r_cm* !E]

Applies the Brindley correction for spherical particles. The macro *Apply_Brindley_Spherical_R_PD* is defined as:

```
macro Apply_Brindley_Spherical_R_PD(& R, & PD) {
    brindley_spherical_r_cm = R PD;
}
```

R is the radius of the particle in cm and PD is the packing density. Here's an example:

```
xdd ...
    str
        Apply_Brindley_Spherical_R_PD(R, PD)
        MVW(0,0,0)
    str
        Apply_Brindley_Spherical_R_PD(R, PD)
        MVW(0,0,0)
```

Use of *phase_MAC* or *MVW* is optional as they are are created as needed. The Brindley correction can be applied to all phases including *xo_ls*. In the case of phases that do not have lattice parameters or sites, definitions of *cell_volume*, *cell_mass* and *phase_MAC* is required for the Brindley correction to work and for *weight_percent*(s) to be calculated. This allows for the incorporation of non-structural phases in quantitative analysis; for example, the following is valid as the necessary information have been included:

```
xo_Is
    Apply_Brindley_Spherical_R_PD(0.002, 0.6)
    MVW(654, 230, 0)
    phase_MAC 200
```

[*capillary_diameter_mm* E]...
    capillary_u_cm_inv E
    [*capillary_convergent_beam*][*capillary_divergent_beam*][*capillary_parallel_beam*]
    [*capillary_focal_length_mm* E]
    [*capillary_xy_n* #]

Examples for the capillary convolution (Coelho & Rowles, 2017) are LAB6-STOE.INP and LAB6-D8.INP as found in the directory TEST_EXAMPLES\CAPILLARY. If using a *str* phase then *capillary_u_cm_inv* can be set to the calculated linear absorption coefficient multiplied by a packing density, for example:

```
prm packing_density 0.31208
capillary_diameter_mm @ 0.57313
    capillary_u_cm_inv
        = Get(mixture_MAC) Get(mixture_density_g_on_cm3) packing_density;
    capillary_focal_length_mm @ 197.89657
    capillary_convergent_beam
```

If not defined, *capillary_focal_length_mm* defaults to the diffractometer radius *Rs*.

[*circles_conv* E]...

Defines ε_m in the convolution function:

$$(1 - \left| \varepsilon_m / \varepsilon \right| \tfrac{1}{2}) \qquad \text{for } \varepsilon = 0 \text{ to } \varepsilon_m$$

that is convoluted into phase peaks. ε_m can be greater than or less than zero. *circles_conv* is used for example by the *Simple_Axial_Model* macro.

[*cloud* $sites]...
    [*cloud_population* !E]
    [*cloud_save* $file]
    [*cloud_save_xyzs* $file]
    [*cloud_load_xyzs* $file]
       [*cloud_load_xyzs_omit_rwps* !E]
    [*cloud_formation_omit_rwps* !E]
    [*cloud_try_accept* !E]
    [*cloud_gauss_fwhm* !E]
    [*cloud_extract_and_save_xyzs* $file]
       [*cloud_number_to_extract* !E]
       [*cloud_atomic_separation* !E]

cloud allows for the tracking of atoms defined in $sites in three dimensions. It can be useful for determining the average positions of heavy atoms or rigid bodies during refinement cycles. For example, a dummy atom, "site X1" say, can be placed at the center of a benzene ring and then tracked as follows:

```
continue_after_convergence
...
cloud "X1"
    cloud_population 100
    cloud_save SOME_FILE.CLD
```

On termination of refinement the CLD file is saved; it can be viewed using the rigid body editor of the GUI; see examples AE14-12.INP for a cloud example. *cloud_population* is the maximum number of population members. Each population member comprises the fractional coordinates of $sites and an associated Rwp value.

cloud_save_xyzs saves a cloud populations to file.

cloud_load_xyzs loads and reuses previously saved populations. *cloud_load_xyzs_omit_rwps* can be used to exclude population membes whilst loading; it can be a function of Get(Cloud_Rwp) where Cloud_Rwp is the associated Rwp of a population member.

cloud_formation_omit_rwps can be used to limit population membes in the formation of CLD files; it can be a function of Get(Cloud_Rwp).

cloud_try_accept accepts population members if it evaluates to non-zero and if the best Rwp since the last acceptance is less than a present population member or if the number of members is less than *cloud_population*. If the number of population members equals *cloud_population* then the population member with the lowest Rwp is discarded. *cloud_try_accept* is evaluated at the end of each refinement cycle; it default value is true. Here's are some examples:

```
cloud_try_accept = And(Cycle, Mod(Cycle, 50));
cloud_try_accept = T == 10;
```

cloud_gauss_fwhm is the full width at half maximum of a three dimensional Gaussian that is used to fill the cloud.

cloud_extract_and_save_xyzs searches the three dimensional cloud for high densities and extracts xyz positions; these are then saved to $file. *cloud_number_to_extract* defines the number of positions to extract and *cloud_atomic_separation* limits the atomic separation during the extraction. The actual number of positions extracted may be less than *cloud_number_to_extract* depending on the cloud.

[*conserve_memory*]

Deletes temporary arrays used in intermediate calculations as often as possible; memory savings of up to 70% can be expected on some problems with subsequent lengthening of execution times by up to 40%. When *approximate_A* is used on dense matrices then *conserve_memory* can reduce memory usage by up to 90%.

[*convolution_step* #1]

An integer defining the number of calculated data points per measured data point. Increasing *convolution_step* when the measurement step is large improves convolution accuracy. Only when the measurement step is greater than about 0.03 degrees 2Th or when high precision is required is it necessary to increase *convolution_step*.

[*default_I_attributes* E]

Changes the attributes of the *I* parameter, for example:

```
    xo_Is...   default_I_attributes 0 min 0.001 val_on_continue 1
```

Useful when randomizing lattice parameters during Le Bail refinements with *continue_after_convergence*.

[*degree_of_crystallinity* #]
    [*crystalline_area* #]
    [*amorphous_area* #]

Reports on the degree of crystallinity which is calculated as follows:

100 * Get(*crystalline_area*) / (Get(*crystalline_area*) + Get(*amorphous_area*))

crystalline_area and *amorphous_area* corresponds to the sum of the numerical areas under the crystalline and amorphous phases respectively. Phases with *amorphous_phase* are treated as amorphous in the calculation.

[*d_Is*]...
    [*d* E *I* E]...

Defines a phase type that uses d-spacing values for generating peak positions. *d* corresponds to the peak position in d-space in Å and *I* is the intensity parameter before applying any *scale_pks* equations.

[*d_spacing_to_energy_in_eV_for_f1_f11* !E]

Can be a function of the reserved parameter *D_spacing*. f′ and f″ (see section 17.6) values used corresponds to energies given by *d_spacing_to_energy_in_eV_for_f1_f11*. Used for refining on energy dispersive data (see example ED_SI_STR.INP), for example:

```
' E(eV) = 10^5 / (8.065541 Lambda(A))
prm !detector_angle_in_radians = 7.77 Deg_on_2;
prm wavelength = 2 D_spacing Sin(detector_angle_in_radians);
prm energy_in_eV = 10^5 / (8.065541 wavelength);
pk_xo = 10^-3 energy_in_eV + zero;
d_spacing_to_energy_in_eV_for_f1_f11 = energy_in_eV;
```

[*exclude* #x1 #x2]...

Excludes an x-axis region between #x1 and #x2. The macro *Exclude* simplifies usage; see example CEO2.INP.

[*exp_conv_const* E [*exp_limit* E]]...

Defines ε_m in the aberration function:

$$A = \text{Exp}(\text{Ln}(0.001)\, \varepsilon / \varepsilon_m) \quad \text{for } \varepsilon = 0 \text{ to } exp\_limit$$

that is convoluted into phase peaks. Used by the *Absorption* and *Absorption_With_Sample_Thickness_mm* macros. The ε range of A is:

$$(0 < \varepsilon < \varepsilon_{limit}) \text{ for } \varepsilon_m < 0, \quad \text{or,} \quad (\varepsilon_{limit} < \varepsilon < 0) \text{ for } \varepsilon_m > 0$$

where $A(\varepsilon_{limit}) = 0.001$. Alternatively ε_{limit} can be defined using *exp_limit*.

[*extra_X_left* !E] [*extra_X_right* !E]

Determines the extra x-axis range for hkl generation. For TOF data *extra_X_left* is typically used. For x-ray data then *extra_X_right* is typically used. Both defaults to 0.5.

[*file_name_for_best_solutions* $file]

Appends INP file details to $file during refinement with independent parameter values updated. The operation is performed when convergence results in the best Rwp.

[*force_positive_fwhm*]

Forces final Lorentzian and/or Gaussian FWHM values to be positive. The following INP snippets are equivalent:

```
force_positive_fwhm                          xdd ...
xdd ...                                          str ...
    str ...                                          lor_fwhm = Abs(Rand(-1.1,));
        lor_fwhm = Rand(-1,1);
```

[*fit_obj* E [*min_X* !E] [*max_X* !E]]...
    [*fo_transform_X* !E]
    [*fit_obj_phase* !E]

Defines User defined functions added to *Ycalc*, see example PVS.INP. *fit_obj*'s can be a function of *X*. *min_X* and *max_X* define the x-axis range of the *fit_obj*; if *min_X* is omitted then the *fit_obj* is calculated from the start of the x-axis; similarly, if *max_X* is omitted then the *fit_obj* is calculated to the end of the x-axis. *fo_transform_X* is a dependent of *fit_obj* and it can be used to transform *X* used within the *fit_obj*.

[*fourier_map* !E]
    [*fourier_map_formula* !E]
    [*extend_calculated_sphere_to* !E]
    [*min_grid_spacing* !E]
    [*correct_for_atomic_scattering_factors* !E]
    [*f_atom_type* $type *f_atom_quantity* !E]...

If *fourier_map* is non-zero then a Fourier map is calculated on refinement termination and shown in the OpenGL window; maps can be calculated for x-ray or neutron single crystal or powder data, see test examples FOURIER-MAP-AE14.INP and FOURIER-MAP-CIME.INP. The type of map is determined by *fourier_map_formula* which can be a function of the reserved parameter names *Fcalc*, *Fobs* and *D_spacing*; here are some examples:

```
fourier_map_formula = Fobs; ' The default
fourier_map_formula = 2 Fobs - Fcalc;
```

For single crystal data, *Fobs* corresponds to the observed structure moduli; powder data *Fobs* is calculated from the Rietveld decomposition formula. Structure factor phases are determined from *Fcalc*. Reflections that are missing from within the Ewald sphere are included with *Fobs* set to *Fcalc*. If *extend_calculated_sphere_to* is defined, then the Ewald sphere is extended. *scale_pks* definitions are removed from *Fobs*. If *scale_pks* evaluates to zero for a particular reflection, then *Fobs* is set to *Fcalc*; the number of Fobs reflections set to *Fcalc* is reported on.

[*gauss_fwhm* E]...

Defines the FWHM of a Gaussian function convoluted into phase peaks; see *CS_G* and *Strain_G* macros.

[*hkl_plane* $hkl]...

Used by the OpenGL viewer to display hkl planes, see the CEO2.STR file in the RIGID directory. Here are some examples:

```
      str ...
         hkl_plane 1 1 1
         hkl_plane "2 -2 0"
```

[*grs_interaction* [*from_N* #] [*to_N* #] [*no_self_interaction*] $site_1 $site_2 qi # qj # N E]...

Defines a GRS interaction with the name N between sites identified by $site_1 and $site_2. E represents the GRS interaction equation that can be a function of *R*; *R* returns the distance in Å between two atoms; these distances are updated when dependent fractional atomic coordinates are modified. The name N of the *grs_interaction* can be used in equations including penalty equations. When either *from_N* or *to_N* are defined, the interactions between $site_1 and $site_2 are sorted by distance and only the interactions between the *from_N* and *to_N* are considered. *no_self_interaction* prevents any interactions between equivalent positions of the same site. This is useful when a general position is used to describe a special position. *qi* and *qj* corresponds to the valence charges used to calculate the Coulomb sum for the $site_1 and $site_2 sites respectively. *grs_interaction* is typically used for applying electrostatic restraints in inorganic materials. The *GRS_Interaction* macro simplifies the use of *grs_interaction*.

[*hat* E [*num_hats* #1]]...

Defines the x-axis extent of an impulse function that is convoluted into phase peaks. *num_hats* correspond to the number of hats to be convoluted. *hat* is used for example by the *Slit_Width* and *Specimen_Tilt* macros.

[*hkl_Is*]...
    [*lp_search* !E]
    [*I_parameter_names_have_hkl* $start_of_parameter_name]
    [*hkl_m_d_th2* # # # # # # *I* E]...

Defines a phase type that uses hkls for generating peak positions. *lp_search* uses a indexing algorithm that is independent of d-spacing extraction ([b]Coelho, 2017); see LP-SEARCH-PBSO4.INP. *lp_search* minimizes on a figure of merit function that gives a measure of correctness for a particular set of lattice parameters. The method avoids difficulties associated with extracting d-spacings from complex patterns comprising heavily overlapped lines; the primary difficulty being that of ascertaining the number of lines present. *I_parameter_names_have_hkl* assigns names to generated Intensity parameters that start with $start_of_parameter_name and end with the corresponding hkl. The numbers after *hkl_m_d_th2* define h k l m d and 2θ values, where

h, k, l	: Miller indices
m	: multiplicity.
d and th2	: d and 2θ values.
I	: Peak intensity parameter before applying any *scale_pks*.

If no *hkl_m_d_th2* keywords are defined, then hkls are generated using the space group. Generated *hkl_m_d_th2* details are placed after the *space_group* keyword on refinement termination. Intensity parameters are given an initial starting value of 1. If *lebail* is not defined, then the intensity parameters are given the code of @. For example, the following:

```
        xdd quartz.xdd
            ...
            hkl_Is
                Hexagonal(4.91459, 5.40603)
                space_group P_31_2_1
```

generates in the OUT file the following:

```
        xdd quartz.xdd
            ...
            hkl_Is
                Hexagonal(4.91459, 5.40603)
                space_group P_31_2_1
                load hkl_m_d_th2 I {
                    1   0   0   6    4.25635    20.85324 @ 3147.83321
                    1   0   1   6    3.34470    26.62997 @ 8559.23955
                    1   0  -1   6    3.34470    26.62997 @ 8559.23955
                    ...
                }
```

The *Create_hklm_d_Th2_lp_file* macro creates an hkl file listing in the "*load hkl_m_d_th2 I*" format as shown above. Even though the structure would have no sites, *weight_percent* can still be used; it uses whatever value is defined by *cell_mass* in order to calculate *weight_percent*.

[*inp_text* $name] ...
[*inp_text_insert* $name { ... }]...

inp_text provides a means of defining INP text at one place in a file and having that text inserted at another place in the INP file, or in an #include file, using *inp_text_insert*. The *inp_text* is updated on refinement termination. *inp_text* is very useful in simplifying complicated INP files where placing control parameters at the top of the file is of benefit; see test_example INP-TEXT.INP. An example is as follows:

```
    inp_text back_ground {
        bkg @  17.365576`  14.5555883`  14.038067`
    }
    xdd …
        inp_text_insert back_ground
```

More than one *inp_text* can be of the same name; in such cases *inp_text_insert* will use the most recent *inp_text*.

[*iters* #]

The maximum number of refinement iterations, default is 500000.

[*lam* [*ymin_on_ymax* #] [*no_th_dependence*] [*Lam* !E] [*calculate_Lam*]]
    [*la* E *lo* E [*lh* E] | [*lg* E] [*lo_ref*]]...

Defines an emission profile (see section 5) where each *la* determines an emission profile line, where:

la: Area under the emission profile line

lo: Wavelength in Å of the emission profile line

lh: HW in mÅ of a Lorentzian convoluted into the emission profile line.

lg: HW in mÅ of a Gaussian convoluted into the emission profile line.

ymin_on_ymax determines the x-axis extent to which an emission profile line is calculated; default value is 0.001. *no_th_dependence* defines an emission profile that is 2θ independent; it allows use of non-X-ray data or fitting to negative 2θ values. By default, the program calculates d-spacings using the wavelength of the emission profile line with the highest *la* parameter. However, if *la* parameters are refined the reference wavelength could change causing confusion. To avoid this *lo_ref* can be used to identify the reference wavelength.

Lam defines the value to be used for the reserved parameter *Lam*. When *Lam* is not defined then the reserved parameter *Lam* is defined as the wavelength of the emission profile line with the largest *la* value. Note that *Lam* is used to determine the Bragg angle.

calculate_Lam calculates *Lam* such that it corresponds to the wavelength at the peak of the emission profile. *Lam* needs to be set to an approximate value corresponding to the peak of the emission profile.

[*lor_fwhm* E]...

Defines the FWHM of a Lorentzian that is convoluted into phase peaks; see for example the *CS_L* and *Strain_L* macros.

[*lpsd_th2_angular_range_degrees* E]...
   lpsd_equitorial_divergence_degrees E
   lpsd_equitorial_sample_length_mm E

Convolutes the aberration for a Linear Position Sensitive Detector (Cheary & Coelho, 1994) into phase peaks. *lpsd_th2_angular_range_degrees* correspond to the angular range of the LPSD in 2Th degrees. *lpsd_equitorial_divergence_degrees* is the equatorial divergence in degrees of the primary beam and *lpsd_equitorial_sample_length_mm* the length of the sample in the equatorial plane. *lpsd_th2_angular_range_degrees* corrects peak shapes, intensities and 2Th shifts, see example LPSD-SIMULATED.INP.

[*min_r* #][*max_r* #]

Defines the minimum and maximum radii for calculating bond lengths, defaults are 0 and 3.2 Å respectively.

[*neutron_data*]

Signals the use of neutron atomic scattering lengths. Scattering lengths for isotopes can be used by placing the isotope name after "*occ*" as in:

```
occ 6Li 1
occ 36Ar 1
```

The scattering lengths data, contained in the file NEUTSCAT.CPP, is from

www.ccp14.ac.uk/ccp/web-mirrors/neutrons/n-scatter/n-lengths/LIST~1.HTM

Constant wavelength neutron diffraction requires a Lorentz correction using the *Lorentz_Factor* macro (defined in TOPAS.INC); it is defined as follows:

```
scale_pks = 1 / (Sin(Th)^2 Cos(Th));
```

[*no_LIMIT_warnings* **]**

Suppresses LIMIT_MIN and LIMIT_MAX warnings.

[*normalize_FCs* **]**

Normalizes site fractional coordinates. Normalization does not occur for coordinates with *min*/*max* limits, or is part of a *rigid* body or is part of a site constraint of any kind.

[*numerical_area* E **]**

Returns the numerically calculated area under the phase.

[*num_cycles* # **]**

Determines the number of cycles to process when *continue_after_convergence* is defined. The number of iterations, defined using *iters*, is still adhered to. Thus, to set number of cycles to 100 then using something like:

```
continue_after_convergence
iters 1000000000
num_cycles 100
```

[*occ_merge* $sites **[** *occ_merge_radius* !E **]]**...

Rewrites site occupancies of sites defined in $sites in terms of their fractional atomic coordinates (Favre-Nicolin and R. Cerny 2002). This is useful during structure solution for merging rigid bodies such as ocathedra. It is also useful for identifying special positions as seen in the example PBSO4-DECOMPOSE.INP. In the present implementation $sites are thought of as spheres with a radius *occ_merge_radius.* When two atoms approach with a distance less than the sum of their respective *occ_merge_radius*'s then the spheres intersect. The occupancies of the sites, occ_xyz, become:

$$occ\_xyz = 1 / (1 + Intersecting\_fractional\_volumes)$$

In this way any number of sites can be merged. Sites appearing in $sites cannot have their occupancies refined. On termination of refinement the *occ* parameter values are updated with their corresponding occ_xyz.

[*omit_hkls* !E **]**

Allows for the filtering of hkls using the reserved parameter names of *H*, *K*, *L* and *D_spacing*. More than one *omit_hkls* can be defined, for example:

```
        omit_hkls = If(And(H==0, K==0), 1, 0);
        omit_hkls = And(H==0, K==1);
        omit_hkls = D_spacing < 1;
```

[*one_on_x_conv* E]...

Defines ε_m in the convolution function:

$$(4 \mid \varepsilon_m \, \varepsilon \mid) - \tfrac{1}{2} \qquad \text{for } \varepsilon = 0 \text{ to } \varepsilon_m$$

that is convoluted into phase peaks. ε_m can be greater than or less than zero, see for example the *Divergence* macro.

[*out_A_matrix* $file]
  [*A_matrix_prm_filter* $filter]

Outputs the least squares **A** matrix to the file $file; used in the macro *Out_for_cf*. Output can be limited by using *A_matrix_prm_filter*, here's an example for outputting **A** matrix elements corresponding to parameters with names starting with 'q':

```
        out_A_matrix file.a A_matrix_prm_filter q*
```

[*out* $file [*append*]]...
  [*out_record*]
    [*out_eqn* !E]
    [*out_fmt* $c_fmt_string]
    [*out_fmt_err* $c_fmt_string]...

Used for writing parameter details to a file. The details are appended to $file when *append* is defined. *out_eqn* defines the equation or parameter to be written to $file using the *out_fmt*. $c_fmt_string describes a format string in c syntax containing a single format specified for a double precision number. *out_fmt_err* defines the $c_fmt_string used for formatting the error of *eqn*. Both *out_fmt* and *out_fmt_err* requires an *out_eqn* definition. *out_fmt* can be used without *out_eqn* for writing strings. The order of *out_fmt* and *out_fmt_err* determines which is written to file first; thus. The following illustrates the use of *out* using the *Out* macros (see OUT-1.INP):

```
        xdd ...
            out "sample output.txt" append
            str ...
                CS_L(cs_l, 1000)
                Out_String("\tCrystallite Size Results:\n")
                Out_String("\t=======================\n")
                Out(cs_l, "\tCrystallite Size (nm):\t%11.5f",
                        "\tError in Crystallite Size:\t%11.5f\n")
```

[*out_rwp* $file]

Outputs a list of R_{wp} values encountered during refinement to the file $file in XDD format.

[*out_prm_vals_per_iteration* $file]... |[*out_prm_vals_on_convergence* $file]...
   [*out_prm_vals_filter* $filter]
   [*out_prm_vals_dependents_filter* $filter_dependents]

Outputs refined independent parameter values per iteration or on convergence into the file $file. *out_prm_vals_filter* can be used to filter the parameters; $fliter can contain the wild card character '*' and the negation character '!', for example:

```
out_prm_vals_per_iteration PRM_VALS.TXT out_prm_vals_filter "* !u*"
```

More than one *out_prm_vals_per_iteration*/*out_prm_vals_on_convergence* can be defined outputting different parameters into different files depending on the corresponding *out_prm_vals_filter*. *out_prm_vals_dependents_filter* allows dependent parameters to be outputted according to $filter_dependents.

[*p1_fractional_to_file* $file][*in_str_format*]

Structure dependent. Saves atomic positions corresponding to space group *P*1 to the file $file. The original space group can be any space group. If *in_str_format* is defined, then the structural data is saved in INP format.

[*peak_type* $type]
   [*pv_lor* E *pv_fwhm* E]
   [*h1* E *h2* E *m1* E *m2* E]
   [*spv_h1* E *spv_h2* E *spv_l1* E *spv_l2* E]

Sets the peak type for a phase, see section 5. The following *peak_type*'s are available:

Peak type	$type	Parameters
Fundamental Parameters	fp	
Pseudo-Voigt	pv	*pv_lor*: the Lorentzian fraction of the peak profile(s). *pv_fwhm*: the FWHM of the peak profile(s).
Split-PearsonVII	spvii	The sum of *h1* and *h2* gives the FWHM of the composite peak. *m1*, *m2* are the PearsonVII exponents of the left and right composite peak.
Split-PseudoVoigt	spv	The sum of *spv_h1* and *spv_h2* gives the full width at half maximum of the composite peak. *spv_l1*, *spv_l2* are the left and right Lorentzian fractions.

[*peak_buffer_step* E [*report_on*]]

Peaks shapes typically change in a gradual manner over a short 2θ range; a new peak shape, therefore, is calculated only if the position of the last peak shape calculated is more than the distance defined by *peak_buffer_step*. Various stretching and interpolation procedures are used to calculate in-between peaks, see also section 5.4. The default setting is as follows:

   peak_buffer_step = 500*Peak_Calculation_Step*.

When the reserved parameter names of *H, K, L, M,* or parameter names associated with *sh_Cij_prm* and *hkl_angle*, are used in peak convolution equations, then irregular peak shapes are possible over short 2θ ranges. In such cases, separate peak shapes are calculated for each peak irrespective of *peak_buffer_step*. *report_on* displays the number of peaks in the peaks buffer.

[*phase_out* $file [*append*]]...

Used for writing phase dependent details to file. See *out* for a description of *out_record*. The *Create_hklm_d_Th2_lp_file* uses *phase_out*.

[*phase_out_X* $file [*append*]]...

Phase dependent keyword that writes phase Ycalc details to a file. The *out_eqn* can contain reserved parameter names occurring in *xdd_out* as well as Get(phase_ycalc); for example:

```
phase_out_X Phase.txt load out_record out_fmt out_eqn {
    " %9.0f" = Xi;
    " %11.5f" = X;
    " %11.5f" = Get(phase_ycalc);
    " %11.5f" = Ycalc;
    " %11.5f" = Yobs;
    " %11.5f\n" = Get(weighting);
}
```

The x-axis extent of the output corresponds to the x-axis range of the phase. If *conserve_memeory* is used, then the message "phase_out_X: No data" is outputted.

[*pk_xo* E]

Applied to all phase types except for *xo_ls* phases; provides a mechanism for transforming peak position to an x-axis position. For example, the peak position for neutron time-of-flight data is typically calculated in time-of-flight space, *tof*, or,

$$tof = t_0 + t_1\, d_{hkl} + t_2\, d_{hkl}{}^2$$

where t_0 and t_1 and t_2 are diffractometer constants. See examples TOF_BALZAR_SH1.INP and TOF_BALZAR_BR1.INP.

[*phase_name* $phase_name]

The name given to a phase; used for reporting purposes.

[*phase_penalties* $sites N]...[*hkl_Re_Im* #h #k #l #Re #Im]...
    [*accumulate_phases_and_save_to_file* $file]
      [*accumulate_phases_when* !E]

phase_penalties for a single hkl is defined as follows:

$$Pp_{hkl} = \begin{cases} 0, & if\ \varphi_{s,hkl} - 45° < \varphi_c < \varphi_{s,hkl} + 45° \\ d\, I^2_{c,hkl}\left(\varphi_{s,hkl} - \varphi_{c,hkl}\right), & if\ \varphi_c < \varphi_{s,hkl} - 45°\, or\ \varphi_c > \varphi_{s,hkl} + 45° \end{cases}$$

where φ_s assigned phase, φ_c = calculated phase, I_c = calculated intensity and *d* is the reflection d-spacing. The name N returns the sum of the *phase_penalties* and it can be used in equations and in particular *penalty* equations. φ_c is calculated from sites identified in $sites.

#h, #k, #l are user defined hkls; they are used for formulating the phase penalties. #Re and #Im are the real and imaginary parts of φ_s. An example usage of phase penalties (see examples AE14-12.INP and AE5-AUTO.INP) is as follows:

```
penalty = pp1;
phase_penalties * pp1
load hkl_Re_Im {
   0   1   2   1   0
   1   0  -2   1   0
   1  -2  -1   1   0
}
```

hkls chosen for phase penalties should comprise those that are of high intensity, large d-spacing and isolated from other peaks to avoid peak overlap. Origin defining hkls are typically chosen.

accumulate_phases_and_save_to_file saves the average phases collected to $file. Phases are collected when *accumulate_phases_when* evaluates to true; *accumulate_phases_when* defaults to true. Here's an example use:

```
load temperature { 1 1 1 1 10 }
    move_to_the_next_temperature_regardless_of_the_change_in_rwp
accumulate_phases_and_save_to_file SOME_FILE.TXT
    accumulate_phases_when = T == 10;
```

Here phases with the best Rwp since the last accumulation are accumulated when the current temperature is 10.

[*process_times*]

Displays process times on termination of refinement.

[*rand_xyz* !E]

If *continue_after_convergence* is defined, then *rand_xyz* is executed at the end of a refinement cycle. It adds the vector **u** to the site fractional coordinate, the direction of which is random and the magnitude in Å is:

|**u**| = *T rand_xyz*

where *T* is the current *temperature*. To add a shift to an atom between 0 and 1 Å the following could be used:

```
temperature 1
    site... occ 1 C beq 1 rand_xyz = Rand(0,1);
```

Only fractional coordinates (*x*, *y*, *z*) that are independent parameters are considered.

[*r_bragg* #]

Reports on the R-Bragg value. R-Bragg is independent of hkl's and thus can be calculated for all phase types that contain phase peaks.

[*rebin_with_dx_of* !E]
   [*rebin_start_x_at* !E]

Rebins the observed data (and *SigmaYobs* if it exists), see example CLAY.INP. It can be a function of the reserved parameter *X* as demonstrated in TOF_BANK2_1.INP. If *rebin_with_dx_of* evaluates to a constant, then the observed data is re-binned to equal x-axis steps. For observed data that is of unequal x-axis steps then re-binning provides a means of converting to equal x-axis steps. Some points about *rebin_with_dx_of*:

- It changes the data.
- It uses all of the data and it uses it once.
- Errors are similar if the fit to the new data is similar.
- If a *hat* convolution is included in *Ycalc* then the fit is potentially the same.

rebin_with_dx_of creates a new x-axis with points determined by the *rebin_with_dx_of* equation, or,

$$x[i+1] = x[i] + \textit{rebin\_with\_dx\_of}$$

The new x-axis can be at x-axis intervals that are unequal. The position of the first x[i] value defaults to the start of the original x-axis; this can be changed using *rebin_start_x_at*. Intensities at the new x-axis are determined by the following integration:

$$\text{Intensity at } x[i] = \text{Integrate } \textit{Yobs} \text{ from } (x[i] + x[i-1])/2 \text{ to } (x[i] + x[i+1])/2$$

Yobs is considered as line segments; integration is therefore simply the area under the line segments. The integration implies convolution and *rebin_with_dx_of* can be thought of as a resampling of the data. If *rebin_with_dx_of* is a constant, then the x-axis intervals are equal, then the integration can be included in *Ycalc* using a hat function, or,

```
rebin_with_dx_of 0.02
hat 0.02
```

[*Rp* #][*Rs* #]

Primary and secondary diffractometer radii in mm; defaults to 217mm.

[*scale* E]

Rietveld scale factor; can be applied to all phase types.

[*scale_pks* E]...

Scales phase peaks; the following defines a Lorentz-Polarisation correction:

```
        scale_pks = (1 + Cos(c Deg)^2 Cos(2 Th)^2) / (Sin(Th)^2 Cos(Th));
```

See *LP_Factor*, *Preferred_Orientation* and *Absorption_With_Sample_Thickness_mm_Intensity* macros.

[*seed* [#]]

Initializes the random number generator with a seed based on the computer clock. To initialize the random number generator at the pre-processor stage then use *#seed*.

[*site* $site [*x* E][*y* E][*z* E]]...
    [*occ* $atom E [*beq* E][*scale_occ* E]]...
    [*num_posns* #][*rand_xyz* !E][*inter* !E #]

Defines a site where $site is a User defined string used to identify the site. *x*, *y*, and *z* define the fractional atomic coordinates, see also section 17.21. *occ* and *beq* defines the site occupancy factor and the equivalent isotropic temperature factor. $atom corresponds to a valid atom symbol or isotope contained in the file ATMSCAT.CPP for x-ray data and NEUTSCAT.CPP for *neutron_data*. *num_posns* corresponds to the number of unique equivalent position generated from the space group; it is updated on termination of refinement. *inter* corresponds to the sum of all GRS interactions which are a function of the *site*. The value of *inter* can represent the site electrostatic potential depending on the type of GRS interactions defined. A site fully occupied by Calcium is written as:

```
        site Al1  x 0        y 0    z 0.3521  occ Ca+2 1    beq 0.3
```

A site occupied by two cations is:

```
        site Fe2  x 0.9283  y 0.25 z 0.9533  occ Fe+3 0.5  beq 0.25
                                             occ Al+3 0.5  beq 0.25
```

scale_occ is *occ* dependent and it scales *occ*. It and can be a function of *H*, *K*, *L*, *D_spacing*, *Xo* and *Th*. The *occ* keyword remains single valued for QUANT purposes and thus cannot be a function of *H*, *K*, *L* etc. The following is valid:

```
        occ Pb+2 1
            prm q1 1 min 1e-6
            prm q2 1 min 1e-6
            prm q3 1 min 1e-6
            prm q4 1 min 1e-6
            scale_occ = q1 / D_spacing + 1 / (q2 H^2 + q3 K^2 + q4 L^2);
```

scale_occ is a child of *occ*, the keyword therefore needs to occur after the *occ* keyword. The following two definitions will produce identical structure factors but different QUANT results:

```
        site Pb occ Pb+2 1    beq 1
        site Pb occ Pb+2 0.5 beq 1 scale_occ 2
```

scale_occ works with magnetic data, neutron data, x-ray data etc. but not PDF data.

Symmetry: The user is responsible for obeying symmetry. If not working in *P*1 then the *Multiplicities_Sum* macro could be used. The *spherical_harmonics_hkl* keyword can also be used, for example:

```
spherical_harmonics_hkl sh sh_order 6
site Pb occ Pb+2 1 beq 1
   prm q 1 min 1e-6
   scale_occ = q sh;
```

[*sites_distance* N]|[*sites_angle* N]|[*sites_flatten* N [*sites_flatten_tol* !E]]...
   [*site_to_restrain* $site [#ep [#n1 #n2 #n3]]]...

When used in equations the name N of *sites_distance* and *sites_angle* returns the distance in Å between two sites and angle in degrees between three sites respectively. The sites considered are defined by *site_to_restrain*. N can be used in penalty equations to restrain bond lengths. N of *sites_flatten* returns a restraint term that decreases as the sites become coplanar; it is defined as follows:

$$\text{sites\_flatten} = \frac{6}{n(n-1)(n-2)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{k=j+1}^{n} \left(\left|b_i \times b_j . b_k\right| - tol\right)^2, if \left|b_i \times b_j . b_k\right| > tol$$

where *tol* corresponds to *sites_flatten_tol*, *n* corresponds to the number of sites defined by *site_to_restrain*, **b** are Cartesian unit length vectors between the sites and the geometric center of the sites.

#eq, #n1, #n2 and #n3 correspond to the site equivalent position and fractional offsets to add to the sites. This is useful if the structure is already known and constraints are required, for example, in the bond length output (see *append_bond_lengths*):

```
Zr1:0  O1:20  0  0 -1  2.08772
       O1:7   0 -1  0  2.08772  89.658
       O1:10  0  0 -1  2.08772  90.342   90.342
       O1:15 -1  0  0  2.08772  180.000  89.658   89.658
       O1:18 -1  0  0  2.08772  90.342   89.658   180.000 90.342

P1:0   O1:4   0  0  0  1.52473
       O1:8   0  0  0  1.52473  112.923
       O1:0   0  0  0  1.52473  112.923  112.923
       O2:0   0  0  0  1.59001  105.749  105.749  105.749
```

Example constraints using macros looks like:

```
Angle_Restrain(O1 P1 O1 8, 112, 112.92311, 0, 0.001)
Angle_Restrain(O1 18 -1 0 0 Zr1 O1 10 0 0 -1, 89, 89.65750, 0, 0.001)
Distance_Restrain(Zr1 O1 20 0 0 -1, 2.08, 2.08772, 0, 1)
```

BENZENE.INP demonstrates the use of the restraint macros *Distance_Restrain*, *Angle_Restrain* and *Flatten*. OpenGL viewing is recommended. Note, for more than ~6 sites then *sites_flatten* becomes computationally expensive.

[*sites_geometry* $Name]...
    [*site_to_restrain* $site [#ep [#n1 #n2 #n3]]]...

Defines a grouping of up to four sites; $Name is the name given to the grouping. The sites that are part of the group is defined using *site_to_restrain*, for example:

```
sites_geometry some_name
    load site_to_restrain { C1 C2 C3 C4 }
```

Three functions, Sites_Geometry_Distance($Name), Sites_Geometry_Angle($Name) and Sites_Geometry_Dihedral_Angle($Name) can be used in equations to obtain the distance between sites C1 and C2, the angle between C1-C2-C3 and the dihedral angle formed between the planes C1-C2-C3 and C2-C3-C4. The convention used are the same as for z-matrices, see example SITES_GEOMETRY_1.INP.

If $Name contains only two sites, then only Sites_Geometry_Distance($Name) can be used. Three sites defined additionally allows the use of Sites_Geometry_Angle($Name) and four sites defined additionally allows the use of Sites_Geometry_Dihedral_Angle($Name).

Examples SITES_GEOMETRY_1.INP And SITES_GEOMETRY_2.INP demonstartes the use of *sites_geometry*.

[*siv_s1_s2* # #]

Defines the s_1 and s_2 integration limits for the spherical interaction volume of the GRS series.

[*smooth* #num_pts_left_right]

Performs a Savitzky-Golay smoothing on the observed data. The smoothing encompasses (2 * #num_pts_left_right + 1) points.

[*spherical_harmonics_hkl* $name]...
    [*sh_Cij_prm* $Yij E]...
    [*sh_alpha* !E]
    [*sh_order* #]

Defines a hkl dependent symmetrized spherical harmonics series (see section 17.23.1) with a name of $name. When $name is used in equations, it returns the value of the associated spherical-harmonics series.

sh_Cij_prm is the spherical harmonics coefficients which can be defined by the User, or, alternatively if there are no coefficients defined then the *sh_Cij_prm* parameters are generated. Only the coefficients allowed by the selection rules of the point group are generated (Järvinen, 1993). At the end of refinement, the generated *sh_Cij_prm* parameters are appended to *sh_order*. This allows for control over the *sh_Cij_prm* parameters in subsequent refinements. $Yij corresponds to valid symmetrized harmonics that has survived symmetrization. It is internally generated when there are no *sh_Cij_prm* parameters defined by the User.

sh_alpha corresponds to the angle in degrees between the polar axis and the scattering vector; *sh_alpha* defaults to zero degrees which is required for symmetric reflection as is the case for Bragg–Brentano geometry.

sh_order corresponds to the order of the spherical harmonic series which are even numbers ranging from 2 to 8 for non-cubic and from 2 to 10 for cubic systems.

The *PO_Spherical_Harmonics* macro simplifies the use of *spherical_harmonics_hkl*. CLAY.INP demonstrates the use of *spherical_harmonics_hkl* for describing anisotropic peak shapes.

[*stacked_hats_conv* [*whole_hat* E [*hat_height* E]]...[*half_hat* E [*hat_height* E]...]...

Defines hat sizes for generating an aberration function comprising a summation of hats. *whole_hat* defines a hat with an x-axis extent of $\pm$*whole_hat*/2. *half_hat* defines a hat with an x-axis range of *half_hat* to zero if *half_hat*<0; or zero to *half_hat* if *half_hat*> 0. *hat_height* defines the height of the hat; it defaults to 1. *stacked_hats* is used for example to describe tube tails using the *Tube_Tails* macro.

[*start_X* !E] [*finish_X* !E]

Defines the start and finish x-axis region to fit to.

[*str* | *dummy_str*]...

Defines a new structure node.

[*str_hkl_angle* N #h #k #l]...

Defines a parameter name N and a vector normal to the plane defined by h, k and l. When the parameter name is used in an equation, it returns angles (in radians) between itself and the normal to the planes defined by hkls; see the *Preferred_Orientation* and *PO_Two_Directions* macros.

[*suspend_writing_to_log_file* #1]

When *num_runs* > 0, then, by default, output to TOPAS.LOG (or TC.LOG if running TC.EXE) is suspended after the first run (Run_Number == 0). *suspend_writing_to_log_file* changes this behaviour.

[*temperature* !E]...
  [*move_to_the_next_temperature_regardless_of_the_change_in_rwp*]
  [*save_values_as_best_after_randomization*]
  [*use_best_values*]

A temperature regime has no affect unless the reserved parameter *T* is used in *val_on_continue* attributes, or, if the temperature dependent keywords *rand_xyz* or *randomize_on_errors* are used. *randomize_on_errors* automatically determine parameter displacements without the need for *rand_xyz* or *val_on_continue*. It performs well on a wide range of problems. The reserved parameter *T* returns the current temperature. The first *temperature* defined becomes the starting temperature; subsequent *temperature*(s) become the current

temperature. If χ_0^2 increases relative to a previous cycle, then the temperature is advanced to the next temperature. If χ_0^2 decreases relative to previous temperatures of lesser values, then the current temperature is rewound to a previous temperature such that its previous is of a greater value. *move_to_the_next_temperature_regardless_of_the_change_in_rwp* forces the refinement to move to the next temperature regardless of the change in Rwp from the previous temperature. *save_values_as_best_after_randomization* saves the current set of parameters and gives them the status of "best solution". Note, this does not change the global "best solution" which is saved at the end of refinement. *use_best_values* replaces the current set of parameters with those marked as "best solution". The temperature regime defined in the macro *Auto_T* is sufficient for most problems. A typical temperature regime starts with a high value and then a series of annealing temperatures, for example:

```
temperature 2
    move_to_the_next_temperature_regardless_of_the_change_in_rwp
temperature 1 temperature 1 temperature 1
```

If the current temperature is the last one defined (the fourth one), and χ_0^2 decreased relative to the second and third *temperature*s, then the current temperature is set to the second temperature. The current temperature can be used in all equations using the reserved parameter *T*, for example:

```
x @ 0.123 val_on_continue = Val + T Rand(-.1, .1)
```

The following temperature regime will allow parameters to randomly walk for the first temperature. At the second temperature the parameters are reset to those that gave the "best solution".

```
temperature 1 temperature 1 use_best_values
temperature 1 temperature 1 use_best_values
temperature 1
temperature 10
    save_values_as_best_after_randomization
    move_to_the_next_temperature_regardless_of_the_change_in_rwp
```

Note, that when a "best solution" is encountered the temperature is rewound to a position where the temperature decreased. For example, if the R_{wp} dropped at lines 2 to 5 then the next temperature will be set to "line 1". The following temperature regime will continuously use the "best solution" before randomisation; it has a tendency to remain in a false minimum:

```
temperature 1 use_best_values
```

[*th2_offset* E]...

Used for applying 2θ corrections to phase peaks. The following applies a sample displacement correction:

```
th2_offset = -2 Rad (c) Cos(Th) / Rs;
```

th2_offset is used for example in the *Zero_Error* and *Specimen_Displacement* macros.

[*user_defined_convolution* E *min* E *max* E]...

User defined convolutions are convoluted into phase peaks and can be a function of *X*. The *min*/*max* equations are mandatory, they define the x-axis extents of the *user_defined_convolution* where min ≤ 0 and max ≥ 0. For example, a sinc function can be convoluted into phase peaks (example AU111.INP) as follows:

```
str ...
    prm k 10 min 0.001 max 100
    user_defined_convolution = If(Abs(X) < 10^(-10), 1, (Sin(k X) /(k X))^2);
        min -3 max 3
```

[*use_tube_dispersion_coefficients*]

Forces the use of Laboratory tube anomalous dispersion coefficients, see section 17.3.

[*verbose* #1]...

A value of 1 instructs the kernel to output in a verbose manner. A value of 0 reduces kernel output such that text output is initiated at the end of a refinement cycle. A value of –1 reduces kernel output such that text output is initiated every second and only R_{wp} values at the end of a refinement cycle is kept. The *Simulated_Annealing_1* macro has *verbose* set to –1; this ensures that long simulated annealing runs do not exhaust memory due to saving R_{wp} values in text output buffers.

[*view_structure*]

Informs a driver GUI to display the structure.

[*weighting* !E [*recal_weighting_on_iter*]]

Used for calculating the *xdd* dependent weighting function in χ_0^2. Can be a function of the reserved parameter names *X*, *Yobs*, *Ycalc* and *SigmaYobs*. The default is as follows:

```
weighting = 1 / Max(Yobs, 1);
```

In cases where *weighting* is a function of *Ycalc* then *recal_weighting_on_iter* can be used to recalculate the weighting at the start of refinement iterations. Otherwise the weighting is recalculated at the start of each refinement cycle. Note that some goodness of fit indicators such as *r_wp* are a function of *weighting*, see Table 4-2.

[*x_calculation_step* !E]

Calculation step used in the generation of phase peaks and *fit_obj*'s. *Peak_Calculation_Step* is the actual step size used. For an x-axis with equidistant steps and *x_calculation_step* not defined then:

Peak_Calculation_Step = "Observed data step size" / *convolution_step*

otherwise

Peak_Calculation_Step = *x_calculation_step* / *convolution_step*

x_calculation_step can be a function of *Xo* and *Th*. In some situations, it may be computation-ally efficient to write *x_calculation_step* in terms of the function Yobs_dx_at and the reserved parameter *Xo*. It is also mandatory to define *x_calculation_step* for data with unequal x-axis steps (*.XY or *.XYE data files). Example usage:

```
x_calculation_step 0.01
x_calculation_step = 0.02 (1 + Tan(Th));
x_calculation_step = Yobs_dx_at(Xo);
```

[*xdd* $file [{ $data }][*range* #][*xye_format*][*gsas_format*][*fullprof_format*]]...
    [*gui_reload*]
    [*gui_ignore*]

Defines the start of *xdd* dependent keywords and the file containing the observed data. {$data} allow for insertion of ASCII data directly into the INP file. *range* applies to Bruker AXS *.RAW data files; in multi-range files it defines the range to be refined with the first range starting at 1 (the default). *xye_format* (see section 17.25 as well) signals the loading of columns of x, y and error values. *gsas_format* and *fullprof_format* signals the loading of GSAS and Full-Prof file formats. The following will refine on the first range in the data file pbso4.raw:

```
xdd pbso4.raw
```

The following will refine on the third range:

```
xdd pbso4.raw range 3
```

To read data directly from an INP file, the following can be used:

```
xdd {
    1 1 10  ' start, step and finish (equidistant data)
    1 2 3 4 5 6 7 8 9 10
}
```

```
xdd {
    _xy ' switch indicating x-y format
    0.1 1   0.2 2   ...
}
```

When in Launch mode; data files by default are not reloaded if already loaded. *gui_reload* forces the reload of the data file. Data files are loaded/reloaded into the GUI under the follow-ing circumstances:

- The data file is not loaded into the GUI
- Any of the following keywords have been used at the *xdd* level:

    gui_reload, *rebin_with_dx_of*, *smooth*, *yobs_eqn*, *yobs_to_xo_posn_yobs*

gui_reload can be used in cases where the data file has been changed by a process not listed. *gui_ignore* informs the GUI to ignore the *xdd* data file; *Ycalc*, difference and other items asso-ciated with the data file is not retrieved from the Kernel.

[*xdd_out* $file [*append*]]...

Used for writing *xdd* dependent details to file. The *out_eqn* can contain the reserved parameter names of *X*, *Yobs*, *Ycalc* and *SigmaYobs*. See *out* for a description of *out_record*. The *Out_Yobs_Ycalc_and_Difference* macro is a good example of using *xdd_out*.

[*xdd_scr* $file]...
　　[*dont_merge_equivalent_reflections*]
　　[*dont_merge_Friedel_pairs*]
　　[*ignore_differences_in_Friedel_pairs*]
　　[*str*]...
　　　　[*auto_scale* !E]
　　　　[*i_on_error_ratio_tolerance* #]
　　　　[*num_highest_I_values_to_keep* #num]

xdd_scr defines single crystal data from the file $file. The file can have extensions of *.HKL for ShelX HKL4 format or *.SCR for SCR format. All *xdd* and *str* keywords that are not dependent on powder data can be used by *xdd_scr*. Single crystal data is internally stored in 2θ versus F_o^2 format; this allows the use of *start_X*, *finish_X* and *exclude* keywords; a *lam* definition is required.

dont_merge_equivalent_reflections prevent merging of equivalent reflections, see also section 17.3.3. *dont_merge_Friedel_pairs* prevent merging of Friedel pairs. *ignore_differences_in_Friedel_pairs* force the use of Eq. (17-12) for calculating F^2. *auto_scale* rewrites the *scale* parameter in terms of F^2; this eliminates the need for the *scale* parameter. The value determined for *auto_scale* is updated at the end of refinement. *i_on_error_ratio_tolerance* filters out hkl's that does not meet the condition:

$$|F_o| > \text{*i\_on\_error\_ratio\_tolerance*} \, |Sigma(F_o)|$$

num_highest_I_values_to_keep removes all hkl's except for #num hkl's with the highest F_o values. An example input segment for single crystal data refinement is as follows:

```
xdd_scr ylidm.hkl
   MoKa2(0.001)
   finish_X 35
   weighting = 1 / (Sin(X Deg / 2) Max(1, Yobs));
   STR(P212121)
      a  5.9636
      b  9.0390
      c  18.3955
      scale @ 1.6039731906
      i_on_error_ratio_tolerance 4
      site S1  x @ 0.8090  y @ 0.1805  z @ 0.7402  occ S 1  beq 2
      site O1  x @ 0.0901  y @ 0.8151  z @ 0.2234  occ O 1  beq 2
      ...
```

The SCR format is white space delimited and consists of entries of h, k, l, m, d, 2θ, F_o^2 which is the format outputted by the *Create_hklm_d_Th2_lp_file* macro.

[*xo_ls*]...

    [*xo* E *I* E]...

Defines a phase type that uses x-axis space for generating peak positions, see example XOIS.INP. *xo* corresponds to the peak position and *I* is the intensity parameter before applying *scale_pks* equations.

[*yobs_eqn* !N E *min* E *max* E *del* E]

Observed data is created via an equation; useful for approximating functions. The name !N given to the equation is used for identifying the plot in the GUI.

[*yobs_to_xo_posn_yobs* !E]

At the start of refinement, *yobs_to_xo_posn_yobs* decomposes an X-ray diffraction pattern into a new pattern comprising at most one data point per hkl. Fitting to the decomposed pattern in a normal Rietveld refinement manner is then possible due to the ability to refine data of unequal x-axis step sizes. This normal Rietveld manner of fitting is important in structure solution from simulated annealing as the background can still be refined and the problem of peak overlap avoided. These new data points are not extracted intensities and thus the problem of peak overlap, as occurs in intensity extraction, is avoided. The much smaller number of data points in the new diffraction pattern can greatly improve speed in structure solution; in otherwords the calculation time in synthesizing the diffraction pattern becomes close to that of when dealing with single crystal data. If the distance between two hkls is less than the value of *yobs_to_xo_posn_yobs* then the proposed data point at one of these hkls is discarded. Thus, the final decomposed pattern may in fact have less data points than hkls. A reasonable value for *yobs_to_xo_posn_yobs* is *Peak_Calculation_Step*, or,

```
yobs_to_xo_posn_yobs = Peak_Calculation_Step;
```

yobs_to_xo_posn_yobs can be a function of the reserved parameter *X* with *X* being the value of the x-axis at the hkl. For refinement stability, all peak shape, zero error and lattice parameters should be determined and then fixed before using *yobs_to_xo_posn_yobs*. Also, if the original diffraction pattern is noisy then it may be best to smooth it using *smooth* or re-binned using *rebin_with_dx_of*. Alternatively, a calculated pattern could be used as input into the *yobs_to_xo_posn_yobs*. Note that structure solution can be speed-up by preventing graphical output or by increasing the Graphics Response Time in the GUI. See examples CIME-DECOM-POSE.INP and PBSO4-DECOMPOSE.INP.

18.3 ... Keywords to simplify User input

18.3.1 ....The "load { }" keyword and attribute equations

"*load* { }" can be used to simplify User input. It allows the loading of keywords of the same type by typing the keywords once, for example, *exclude* in the following input segment:

```
xdd    exclude 20 22    exclude 32 35    exclude 45 47
```

can be rewritten using "*load* { }" as follows:

```
    xdd   load exclude { 20 22 32 35 45 47 }
```

This input can be further simplified using the *Exclude* macro:

```
    xdd   Exclude { 20 22 32 35 45 47 }
```

In some cases, attribute equations are loaded by the parameter itself. For example, in the following:

```
    prm t 0.01  val_on_continue = Rand(-Pi, Pi); min 0.4 max 0.5
```

the *prm* will load the attribute. But, in the following:

```
    load prm val_on_continue min max { t 0.01 = Rand(-Pi, Pi); 0.4 0.5 }
```

load will load the attributes. The following is valid:

```
    load sh_Cij_prm {
        y00 !sh_c00 1
        y20  sh_c20 0.26202642  min 0 max 1
        y40  sh_c40 0.06823548
        ...
    }
```

In this case *load* does not contain *min*/*max* and the parameter will load its attributes. *load* however can contain attributes, for example:

```
    load sh_Cij_prm min max { ' load loads the attributes
        y00 !sh_c00 1            0 1
        y20  sh_c20 0.26202642  0 1
        y40  sh_c40 0.06823548  0 1
        ...
    }
```

18.3.2 ....The "move_to $keyword" keyword

move_to provides a means of entering parameter attributes without having to first load the parameter, see *Keep_Atom_Within_Box* macro. The site dependent *ADPs_Keep_PD* macro defines *min*/*max* limits; here's part of that macro:

```
    move_to u12
        min = -Sqrt(Get(u11) Get(u22));
        max =  Sqrt(Get(u11) Get(u22));
```

$keyword of *move_to* can be any keyword and not just a parameter keyword.

18.3.3 ....The "for xdds { }" and "for strs { }" constructs

The "*for xdds* { }" and "*for strs* { }" constructs simplify the construction of input files when there are multiple diffraction patterns with similar structures, see example QUARTZ2_FPA.INP. For example, two diffraction patterns with the same structures can be composed as follows:

```
xdd ...                    ' first xdd
   bkg ...                 ' xdd dependent, not common to the xdd's
   th2_offset ...
   str ...                 ' first str
      scale ...            ' str dependent, not common to the str's
xdd ...                    ' second xdd
   bkg...                  ' xdd dependent, not common to the xdd's
   th2_offset ...
   str ...                 ' second str
      scale ...            ' str dependent, not common to the str's
for xdds {
   Slit_Width(0.2)         ' xdd dependent, common to the xdd's
   CuKa2(0.0001)
   for strs { ... }        ' str dependent, common to the xdd's
   for strs 1 to 1 {
      space_group p1       ' str dependent, specific to the first str
      site ...
   }
}
```

The effect of using parameter names of @ in *for* constructs is to give unique parameter names to the parameters iterated over; the output file would contain the parameter value corresponding to the last "*for xdds*" or "*for strs*" iteration. Parameters should therefore not be given the @ name within *for* constructs.

19. MACROS AND INCLUDE FILES

Macros appearing in INP files are expanded by the pre-processor. The pre-processor comprises directives of two types, global types and types that are invoked on macro expansion; directives begin with the character # and are:

Directives with global scope:

macro $user_defined_macro_name { ... }

#include $user_defined_macro_file_name

#delete_macros { $macros_to_be_deleted }

#define, *#undef*, *#if*, *#ifdef*, *#ifndef*, *#else*, *#elseif*, *#endif*, *#prm*

#seed – initializes the random number generator at the pre-processor stage.

Directives invoked on macro expansion:

#m_if, *#m_ifarg*, *#m_elseif*, *#m_else*, *#m_endif*

#m_code, *#m_eqn*, *#m_code_refine*, *#m_one_word*

#m_argu, *#m_first_word*, *#m_unique_not_refine*

19.1 ... The macro directive

Macros are defined using the *macro* directive; here's an example:

```
macro Cubic(cv) { a cv b = Get(a); c = Get(a); }
```

Macros can have multiple arguments or none; the *Cubic* macro above has one argument; here are some example uses of *Cubic*:

```
Cubic(4.50671)
Cubic(a_lp 4.50671  min 4.49671  max 4.52671)
Cubic(!a_lp 4.50671)
```

The first instance defines the *a*, *b* and *c* lattice parameters without a parameter name. The second defines the lattice parameters with a name indicating refinement of the *a_lp* parameter. In the third example, the *a_lp* parameter is preceded by the character !. This indicates that the *a_lp* parameter is not to be refined; it can however be used in equations. The definition of macros need not precede its use. For example, in the segment:

```
xdd...
   Emission_Profile ' this is expanded
   macro Emission_Profile { CuKa2(0.001) }
```

Even though the macro *Emission_Profile* has been defined after its use, *Emission_Profile* is expanded to "CuKa2(0.001)".

Macro names need not be unique; in cases where more than one macro have the same name then the actual macro expanded is determined by the number of arguments. For example, if the macro *Slit_Width*(0.1) is used then the *Slit_Width*(v) macro is expanded. On the other hand if the macro

Slit_Width(sw, .1) is used then the *Slit_Width*(c, v) macro is expanded. Macros can also expand to macro names. For example, the macro *Crystallite_Size* expands to *CS* and since *CS* is a macro then the *CS* macro will be expanded.

19.1.1 ....Directives with global scope

#include $user_defined_macro_file_name

Include files are used to group macros. The file TOPAS.INC contains standard macros; its a good place to view examples. Text within include files are inserted at the position of the *#include* directive, thus the following:

```
#include "my include file.inc"
```

inserts the text within "my include file.inc" at the position of the *#include* directive. The standard macro file TOPAS.INC is always included by default.

#delete_macros { $macros_to_be_deleted }

Macros can be deleted using *#delete_macros*, for example the following

```
#delete_macros { LP_Factor SW ZE }
```

will delete previously defined macors, irrespective of the number of arguments, with the names *LP_Factor*, *SW* and *ZE*.

#define, *#undef*, *#ifdef*, *#ifndef*, *#else*, *#endif*

The *#define* and *#undef* directives works similar-to the c pre-processor directives of the same name. *#define* and *#undef* is typically used with *#ifdef*, *#else*, *#endif* directives to control macro expansion in INP files. For example, the following:

```
#ifdef STANDARD_MACROS
    xdd ...
#endif
```

will expand to contain the *xdd* keyword if STANDARD_MACROS has been previously defined using a *#define* directive. The following will also expand to contain the *xdd* keyword if STANDARD_MACROS has not been defined using a *#define* directive,

```
#ifdef !STANDARD_MACROS
    #define STANDARD_MACROS
    xdd ...
#endif
```

or,

```
#ifndef STANDARD_MACROS
    #define STANDARD_MACROS
    xdd ...
#endif
```

Note the use of the '!' character placed before STANDARD_MACROS which means if STAND-ARD_MACROS is not defined.

19.1.2 ....Pre-processor equations and #prm, #if, #elseif, #out

Pre-processor parameters, called hash parameters, are defined using the *#prm* directive. *#prm*'s can be a function of other *#prm*'s and they can be used in *#if*, *#elseif*, *#m_if* and *#m_elseif* pre-processor statements. *#prm*'s are only evaluated at the pre-processor stage of loading INP files (see TEST_EXAMPLES\HASH_PRM.INP); they are therefore unknown to the kernel and are totally separate to parameters defined using *prm*. Pre-processed output can be found in the TOPAS.LOG file, when running TA.EXE, or, TC.LOG when running TC.EXE.

#out and *#m_out* allows pre-processor *#prm*'s values, which can be strings or numbers, to be placed into the pre-processed text. For example, the following:

```
#prm a = Constant(Rand(0,1));
#out a
```

will output a random number between 0 and 1 into the pre-processed file at the position of *#out*. INP files can therefore be manipulated with *#prm*'s and *#if* statements with a means of identifying the manipulation carried out. The following:

```
macro Ex1(a) {
    #m_if a == "b";
        Yes b
    #m_elseif a == "c";
        Yes c
    #m_endif
}
Ex1("b")
```

expands to:

```
    Yes b
```

In the following:

```
#prm ran = Constant(Rand(0,1));
#if ran < 0.5;
    view_structure
#endif
#if ran < 0.5;
    view_structure
#endif
#if ran < 0.5;
    view_structure
#endif
```

each call to 'ran' in the *#if* statements would return the same value because of the use of Constant. More complicated INP file manipulation is shown in the following:

```
#prm space_group_number = 4;
```

```
    #if And(space_group_number >= 75, space_group_number <= 142);
      ...
    #elseif And(space_group_number >= 16, space_group_number <= 74);
      ...
    #endif
```

19.1.3 ....Directives invoked on macro expansion

#m_if, *#m_ifarg*, *#m_elseif*, *#m_else*, *#m_endif*, *#m_if*, *#m_out*

These are conditional directives that are invoked on macro expansion. *#m_ifarg* operates on two statements immediately following its use; the first must refer to a macro argument and the second can be any of the following: *#m_code*, *#m_eqn*, *#m_code_refine*, *#m_one_word* and "some string". *#m_ifarg* evaluates to true according to the rules of Table 19-1.

Table 19-1. *#m_ifarg* syntax and meaning.	
	Evaluates to true if the following is true
#m_ifarg c *#m_code*	If the macro argument c has a letter or the character ! as the first character and if it is not an equation.
#m_ifarg c *#m_eqn*	If the macro argument c is an equation.
#m_ifarg c *#m_code_refine*	If the macro argument c has a letter as the first character and if it is not an equation.
#m_ifarg c "some_string"	If the macro argument c == "some_string".
#m_ifarg v *#m_one_word*	If the macro argument v consists of one word.

#m_argu, *#m_first_word*, *#m_unique_not_refine*

These operate on one macro argument with the intention of changing the value of the argument according to the rules of Table 19-2.

Table 19-2. Directives that change the value of a macro argument.	
#m_argu c	Changes the macro argument c to a unique parameter name if it has @ as the first character.
#m_unique_not_refine c	Changes the macro argument c to a unique parameter name that is not to be refined.
#m_first_word $v	Replace the string macro argument $v with the first word occurring in $v.

19.1.4 ....Defining unique parameters within macros

#m_unique $string assigns a unique parameter name to $string within a macro. This allows new unique parameters to be defined within macros whilst avoiding name clashes. In the example:

```
    macro Some_macro(v) { prm #m_unique a = Cos(Th); : v }
```

'a' is assigned a unique parameter name and it has the scope of the macro body text. The *Robust_Refinement* and *TCHZ_Peak_Type* macros are good examples of its use, where for example, the former is defined as:

```
macro Robust_Refinement {
    ' Robust refinement algorithm
    prm #m_unique test = Get(r_exp);
    prm #m_unique N = 1 / test^2;
    prm #m_unique p0 = 0.40007404;
    prm #m_unique p1 = -2.5949286;
    prm #m_unique p2 = 4.3513542;
    prm #m_unique p3 = -1.7400101;
    prm #m_unique p4 = 3.6140845e-1;
    prm #m_unique p5 = -4.45247609e-2;
    prm #m_unique p6 = 3.5986364e-3;
    prm #m_unique p7 = -1.8328008e-4;
    prm #m_unique p8 = 5.7937184e-6;
    prm #m_unique p9 = -1.035303e-7;
    prm #m_unique p10 = 7.9903166e-10;
    prm #m_unique t = (Yobs - Ycalc) / SigmaYobs;
    weighting = If( t < 0.8,    N / Max(SigmaYobs^2, 1),    If( t < 21, N
                ((((((((((p10 t + p9) t + p8) t + p7) t + p6) t + p5) t + p4) t + p3)
                t + p2) t + p1) t + p0) / (Yobs - Ycalc)^2, N (2.0131 Ln(t) + 3.9183)
                / (Yobs - Ycalc)^2));
    recal_weighting_on_iter
}
```

19.1.5 ....Superfluous parentheses and the '&' Type for macros

The pre-processor is an un-typed language meaning that it knows nothing about the type of text passed to macro arguments. This has great flexibility but care must be take; for example, the following:

```
macro divide(a, b) { a / b }
prm e = divide(a + b, c - d);
```

expands to the unintended result of:

```
prm e = a + b / c - d;
```

The writer of the macro could solve this problem by rewriting the macro with parentheses:

```
macro divide(a, b) { (a) / (b) }
```

Alternatively, macro arguments can be prefixed with the & character signalling that the argument is of an equation type, for example:

```
macro divide(& a, & b) { a / b }
prm e = divide(a + b, c - d);
```

The program inspects &-type arguments and parentheses are included as needed. This results in the correct expansion of:

```
    prm e = (a + b) / (c - d);
```

Even with the use of &-types for arguments, the following:

```
    macro divide(& a, & b) { a / b }
    prm e = divide(a + b, c - d)^2;
```

expands to the unintended:

```
    prm e = (a + b) / (c - d)^2;
```

The writer of the macro could again rewrite the macro to include more parentheses:

```
    macro divide(a, b) {  ((a) / (b))  }
```

Or, define the expansion of the macro itself to have an &-type by placing the & character before the macro name itself:

```
    macro & divide(& a, & b) {  a / b  }
```

Expansion of prm e = divide(a + b, c - d)^2 now becomes the intended:

```
    prm e = ((a + b) / (c - d))^2;
```

With the use of the &-type, macros such as *Ramp* defined in Version 4 as:

```
    macro Ramp(x1, x2, n) { ((x1)+((x2)-(x1)) Mod(Cycle_Iter,(n))/((n)-1)) }
```

can be written with less parentheses as follows:

```
    macro & Ramp(& x1,& x2,& n) { x1 + (x2 - x1) Mod(Cycle_Iter, n) / (n-1) }
```

19.2 ... Overview

The file TOPAS.INC is included in INP files by default; it contains commonly used standard macros. The meaning of the macro arguments in TOPAS.INC can be readily determined from the following conventions:

> Arguments called "c" correspond to a parameter name.
> Arguments called "v" correspond to a parameter value.
> Arguments called "cv" correspond to a parameter name and/or value.

For example, the macro *Cubic*(cv) requires a value and/or a parameter name as an argument, i.e.

```
    Cubic(a_lp 10.604)
    Cubic(10.604)
    Cubic(@ 10.604  min 10.59 max 10.61)
```

Here are examples for the *Slit_Width* macro:

```
SW(@, 0.1)
SW(sw, 0.1  min = Val-.02; max = Val+.02;)
SW((ap+bp)/cp, 0) ' where ap, bp and cp are parameters defined elsewhere
```

19.2.1 ....xdd macros

```
RAW(path_no_ext)
RAW(path_no_ext, range_num)
DAT(path_no_ext)
XDD(path_no_ext)
XY(path_no_ext, calc_step)
XYE(path_ext)
SCR(path_no_ext)
SHELX_HKL4(path_no_ext)
```

19.2.2 ....Lattice parameters

```
Cubic(cv)
Tetragonal(a_cv, c_cv)
Hexagonal(a_cv, c_cv)
Rhombohedral(a_cv, al_cv)
```

19.2.3 ....Emission profile macros

```
No_Th_Dependence          FeKa7_Holzer(yminymax)
CuKa1(yminymax)           MnKa7_Holzer(yminymax)
CuK1sharp(yminymax)       NiKa5_Holzer(yminymax)
CuKa2_analyt(yminymax)    MoKa2(yminymax)
CuKa2(yminymax)           CuKb4_Holzer(yminymax)
CuKa4_Holzer(yminymax)    CoKb6_Holzer(yminymax)
CuKa5(yminymax)           CrKb5_Holzer(yminymax)
CuKa5_Berger(yminymax)    FeKb4_Holzer(yminymax)
CoKa3(yminymax)           MnKb5_Holzer(yminymax)
CoKa7_Holzer(yminymax)    NiKb4_Holzer(yminymax)
CrKa7_Holzer(yminymax)
```

19.2.4 ....Instrument and instrument convolutions

Radius(rp, rs)

Primary and secondary instrument radii (mm). For most diffractometers rp = rs.

Specimen_Tilt(c, v)

Specimen tilt in mm.

Slit_Width(c, v) or SW(c, v)

Aperture of the receiving slit in the equatorial plane in mm.

Sample_Thickness(dc, dv)

Sample thickness in mm in the direction of the scattering vector.

Divergence(c, v)

Horizontal divergence of the beam in degrees in the equatorial plane.

Variable_Divergence(c, v)
Variable_Divergence_Shape(c, v)
Variable_Divergence_Intensity

Constant illuminated sample length in mm for variable slits (i.e. variable beam divergence). This *Variable_Divergence* macro applies both a shape and intensity correction.

Simple_Axial_Model(c, v)

Receiving slit length mm for describing peak asymmetry due to axial divergence.

Full_Axial_Model(filament_cv, sample_cv, detector_cv, psol_cv, ssol_cv)

Accurate model for describing peak asymmetry due to axial divergence of the beam.

[filament_cv]: Tube filament length in [mm].

[sample_cv]: Sample length in axial direction in [mm].

[detector_cv]: Length of the detector (= receiving) slit in [mm].

[psol_cv, ssol_cv]: Aperture of the primary and secondary Soller slit in [°].

Finger_et_al(s2, h2)

Finger *et al.*, 1994. model for describing peak asymmetry due to axial divergence.

[s2, h2]: Sample length, receiving slit length.

Tube_Tails(source_width_c, source_width_v, z1_c, z1_v, z2_c, z2_v, 1z2_h_c, z1z2_h_v)

Model for description of tube tails (Bergmann, 2000).

[source_width_c, source_width_v]: Tube filament width in [mm].

[z1_c, z1_v]: Effective width of tube tails in the equatorial plane perpendicular to the X-ray beam - negative z-direction [mm].

[z2_c, z2_v]: Effective width of tube tails in the equatorial plane perpendicular to the X-ray beam - positive z-direction [mm].

[z1_z2_h_c, z1_z2_h_v]: Fractional height of the tube tails relative to the main beam.

UVW(u, uv, v, vv, w, wv)

Cagliotti relation (Cagliotti *et al.*, 1958).

[u, v, w]: Parameter names.

[uv, vv, wv]: Halfwidth value.

19.2.5 ....Phase peak_type's

PV_Peak_Type(ha, hav, hb, hbv, hc, hcv, lora, lorav, lorb, lorbv, lorc, lorcv)
TCHZ_Peak_Type(u, uv, v, vv, w, wv, x, xv, y, yv, z, zv)
PVII_Peak_Type(ha, hav, hb, hbv, hc, hcv, ma, mav, mb, mbv, mc, mcv)

Pseudo-Voigt, TCHZ pseudo-Voigt and PearsonVII functions. For the definition of the functions and function parameters refer to section 5.2.

19.2.6 ....Quantitative Analysis

Apply_Brindley_Spherical_R_PD(R, PD)

Applies the Brindley correction for quantitative analysis (Brindley, 1945).

MVW(m_v, v_v, w_v)

Returns cell mass, cell volume and weight percent.

19.2.7 ....2Th Corrections

Zero_Error or ZE(c, v)

Zero point error.

Specimen_Displacement(c, v) or SD(c, v)

Specimen displacement error.

19.2.8 ....Intensity Corrections

LP_Factor(c, v)

Lorentz and Lorentz-Polarisation factor.

[c, v]: Monochromator angle in [°2θ]

Values for most common monochromators (Cu radiation) are:

Ge	: 27.3°
Graphite	: 26.4°
Quartz	: 26.6°

Lorentz_Factor

Lorentz factor for fixed wavelength neutron data.

Surface_Roughness_Pitschke_et_al(a1c, a1v, a2c, a2v)
Surface_Roughness_Suortti(a1c, a1v, a2c, a2v)

Suortti and Pitschke *et al.* intensity corrections each with two parameters a1 and a2.

Preferred_Orientation(c, v, ang, hkl) or PO(c, v, ang, hkl)

Preferred orientation correction based on March (1932).

[c, v]: March parameter value.

[ang, hkl]: Lattice direction.

PO_Two_Directions(c1, v1, ang1, hkl1, c2, v2, ang2, hkl2, w1c, w1v)

Preferred orientation correction based on March (1932) considering two preferred orientation directions.

[c1, v1]: March parameter value for the first preferred orientation direction.

[ang1, hkl1]: Parameter name and lattice plane for the first preferred orientation direction.

[c2, v2]: March parameter value for the second preferred orientation direction.

[ang2, hkl2]: Lattice direction for the second preferred orientation direction.

[w1c, w1v]: Fraction of crystals oriented into first preferred orientation direction.

PO_Spherical_Harmonics(sh, order)

Preferred orientation correction based on spherical harmonics according to Järvinen (1993).

[(sh, order)]: Parameter name, spherical harmonics order.

19.2.9 ....Bondlength penalty functions

Anti_Bump(ton, s1, s2, ro, wby)
AI_Anti_Bump(s1, s2, ro, wby, num_cycle_iters), AI_Anti_Bump(s1, s2, ro, wby)

Applies a penalty function as a function of the distance between atoms. The closer the atoms are the higher the penalty is.

[ton]: Sets *to_N* of *box_interaction*.

[s1, s2]: Sites.

[ro]: Distance.

[wby]: Relative weighting given to the penalty function.

For more details refer to *box_interaction* and *ai_anti_bump*.

Parabola_N(n1, n2, s1, s2, ro, wby)

Applies a penalty function as a function of the distance between atoms. The closer the atoms are the higher the penalty is.

[n1]: The closest n1 number of atoms of type s2 is soft constrained to a distance ro away from s1.

[n2]: The closest n2 number of atoms of type s2 (excluding the closest n1 number of atoms of type s2) is repelled from s1, for distances between s1 and s2 less than ro.

[s1, s2]: Sites.

[ro]: Distance.

[wby]: Relative weighting given to the penalty function.

Grs_Interaction(s1, s2, wqi, wqj, c, ro, n)

Penalty function applying the GRS series according to Coelho & Cheary (1997).

[s1, s2]: Sites.

[wqi, wqj]: Valence charge of the atoms.

[c]: Name of the GRS.

[ro]: Distance.

[n]: The exponent of the repulsion part of the Lennard-Jones potential.

For more details refer to *grs_interaction*.

Grs_No_Repulsion(s1, s2, wqi, wqj, c)

Used for calculating the Madelung constants.

[s1, s2]: Sites.

[wqi, wqj]: Valence charge of the atoms.

[c]: Name of the GRS.

Grs_BornMayer(s1, s2, wqi, wqj, c, ro, b)

Uses the GRS series with a Born-Mayer equation for the repulsion term.

[s1, s2]: Sites.

[wqi, wqj]: Valence charge of the atoms.

[c]: Name of the GRS.

[ro]: Mean distance.

[b]: b-constant for the repulsion part of the Born-Mayer potential.

Distance_Restrain(sites, t, t_calc, tol, wscale)
Angle_Restrain(sites, t, t_calc, tol, wscale)
Flatten(sites, t_calc, tol, wscale)
Distance_Restrain_Keep_Within(sites, r, wby, num_cycle_iters)
Distance_Restrain_Keep_Out(sites, r, wby, num_cycle_iters)

Applies penalties restraining distances and angles between sites. 'sites' must comprise two sites for the distance restraints and three for the angle restraints. For Flatten, 'sites' must contain more than three sites. wby is a scaling constant applied to the penalty.

Keep_Atom_Within_Box(size).

Applies a *min*/*max* constraints such that the present site cannot more outside of a box with a length of 2*size.

19.2.10......Reporting macros

Create_2Th_Ip_file(file)

Creates a file with positions (2θ) and intensities.

Create_d_Ip_file(file)

Creates a file with positions (d) and intensities.

Create_hklm_d_Th2_Ip_file(file)

Creates a file with the following information for each peak: h, k, l, multiplicity, positions d and 2θ and intensities.

Out_Yobs_Ycalc_and_Difference(file)

Outputs the x-axis, *Yobs*, *Ycalc* and difference.

Out_X_Yobs(file), Out_X_Ycalc(file), Out_X_Difference(file)

Outputs the x-axis, *Yobs*, *Ycalc* and difference to files.

Out_F2_Details(file), Out_A01_A11_B01_B11(file)

Outputs structure factor details, see section 17.3.2.

Out_FCF(file)

Outputs a CIF file representation of structure factor details suitable for generating Fourier maps using ShelX..

Out_CIF_STR(file)

Outputs structure details in CIF format.

Absorption_With_Sample_Thickness_mm_Shape_Intensity(u, uv, d, dv)

Corrects the peak intensity for absorption effects.

[u, uv]: Parameter name, absorption coefficient in cm-1.

[d, dv]: Parameter name, sample thickness in [mm].

CS_L(c,v) or Crystallite_Size(c, v) or CS(c, v)

Applies a Lorentzian convolution with a FWHM that varies according to the relation lor_fwhm = 0.1 Rad Lam / (c Cos(*Th*)).

[c, v]: Parameter name, crystallite size in [nm].

CS_G(c, v)

Applies a Gaussian convolution with a FWHM that varies according to the relation

```
gauss_fwhm = 0.1 Rad Lam / (c Cos(Th));
```

[c, v]: Parameter name, crystallite size in [nm].

Strain_L(c, v) or Microstrain(c, v) or MS(c, v)

Applies a Lorentzian convolution with a FWHM that varies according to the relation lor_fwhm = c Tan(*Th*).

Strain_G(c, v)

Applies a Gaussian convolution with a FWHM that varies according to the relation gauss_fwhm = c Tan(*Th*).

LVol_FWHM_CS_G_L(k, lvol, kf, lvolf, csgc, csgv, cslc, cslv)

Calculates FWHM and IB (integral breadth) based volume-weighted column heights (LVol). For details refer to section 17.7.

[k, lvol]: shape factor (fixed to 1), integral breadth based LVol.

[kf, lvolf]: shape factor (defaults to 0.89), FWHM based LVol.

[csgc, csgv]: Parameter name, Gaussian component.

[cslc, cslv]: Parameter name, Lorentzian component.

19.2.11......Neutron TOF

TOF_XYE(path, calc_step), TOF_GSAS(path, calc_step)

Includes the *neutron_data* keyword and the calculation step size.

TOF_LAM(w_ymin_on_ymax)

Defines a simple emission profile suitable for TOF data

TOF_x_axis_calibration(t0, t0v, t1, t1v, t2, t2v)

Writes the *pk_xo* equation in terms of the three calibration constants t0, t1, t2 converting d-spacing to x-axis space.

TOF_Exponential(a0, a0v, a1, a1v, wexp, t1, lr)

An exponential convolution applied to the TOF peaks - see example TOF_BANK2_1.INP.

TOF_CS_L(c, v, t1), TOF_CS_G(c, v, t1)

Lorentzian and Gaussian components for crystallite size. t1 is the calibration constant appearing in the argument of the macro *TOF_x_axis_calibration*.

TOF_PV(fwhm, fwhmv, lor, lorv, t1)

A pseudo-Voigt used to describe the instrumental broadening t1 is the calibration constant appearing in the argument of the macro *TOF_x_axis_calibration*, see examples TOF_BALZAR_BR1.INP and TOF_BALZAR_SH1.INP.

19.2.12......Miscalleneous

Temperature_Regime

Defines a temperature regime. See the temperature keyword.

STR(sg)

Signals the start of structure information with a space group of sg.

Exclude

Defines excluded regions. See *exclude*.

Decompose(diff_toll)

Decompose a diffraction pattern comprising data points at peak positions only. Data points closer than diff_toll to another data point is not included. Decompose also sets *x_calculation_step* to the value of diff_toll.

ADPs_Keep_PD

Mixture_LAC_1_on_cm(mlac)

Phase_Density_g_on_cm3(pd)

Phase_LAC_1_on_cm(u)

Gauss(xo, fwhm), Lorentzian(xo, fwhm)

>   An equation defines a unit area Gaussian or Lorentzian with a position of xo and a FWHM of fwhm

20. CHARGE-FLIPPING

The charge-flipping method of Oszlányi & Süto (2004) has been implemented (Coelho, 2007) using the keywords shown in Table 20-2. Also included is the use of the tangent formula (Hauptman & Karle, 1956) within the iterative charge-flipping process. Equations appearing in charge-flipping keywords can be functions of the items shown in Table 20-1. At the end of a charge flipping process a file with the same name as that given by *cf_hkl_file* is created but with a *.FC extension. Almost all charge-flipping keywords can be equations allowing for great flexibility in-regards to changing resolution etc... on the fly. Table 20-3 lists charge-flipping examples found in the CF directory.

Table 20-1. Items that can be used in charge-flipping equations	
Get(Aij) Get(alpha_sum) Get(density) Get(cycles_since_last_best) Get(d_squared_inverse) Get(initial_phase) Get(iters_since_last_best) Get(F000) Get(max_density) Get(max_density_at_cycle_iter_0) Get(num_reflections_above_d_min) Get(phase_difference) Get(r_factor_1), Get(r_factor_2) Get(threshold)	These are updated internally each charge-flipping iteration or cycle or when needed.
Reserved parameter names: *Cycle_Iter*, *Cycle*, *Iter*, *D_spacing*	
Macros (see TOPAS.INC) *Ramp*, *Ramp_Clamp*, *Cycle_Ramp*, *Tangent*, *Restart_CF*, *Pick*, *Pick_Best* *Out_for_cf*(file) : Outputs the A matrix from a Pawley refinement for use in charge flipping; uses *cf_in_A_matrix*. See example CF-CIME-PAWLEY.INP.	

charge_flipping	**Default**
[*a* !E *b* !E *c* !E [*al* !E] [*be* !E] [*ga* !E]]	*al* = *be* = *ga* = 90
[*cf_hkl_file* $file]	
[*cf_in_A_matrix* $file]	
[*scale_Aij* !E]	Get(Aij)^2
[*break_cycle_if_true* !E]	
[*delete_observed_reflections* !E]	
[*extend_calculated_sphere_to* !E]	
[*f_atom_type* $type *f_atom_quantity* !E]...	
[*find_origin* !E]	1

Table 20-2. Keywords that can be used in charge-flipping.

[*fraction_density_to_flip* !E]	0.75
[*fraction_reflections_weak* !E]	0
[*min_d* !E]	0
[*min_grid_spacing* !E]	
[*neutron_data*]	
[*space_group* $]	*P1*
[*use_Fc*]	
Electron density perturbations	
[*flip_equation* !E]	
[*flip_regime_2* !E]	
[*flip_regime_3* !E]	
[*histogram_match_scale_fwhm* !E]	
[*hm_size_limit_in_fwhm* !E]	1
[*hm_covalent_fwhm* !E]	1
[*pick_atoms* $atoms]*...*	
[*activate* !E]	1
[*choose_from* !E]	
[*choose_to* !E]	
[*choose_randomly* !E]	
[*omit* !E]	
[*displace* !E]	
[*insert* !E]	
[*scale_density_below_threshold* !E]	
[*symmetry_obey_0_to_1* !E]	
Phase perturbations	
[*add_to_phases_of_weak_reflections* !E]	
[*randomize_phases_on_new_cycle_by* !E]	0
[*set_initial_phases_to* $file]	
[*modify_initial_phases* !E]	
[*tangent_num_h_read* !E]	
[*tangent_num_k_read* !E]	
[*tangent_num_h_keep* !E]	
[*tangent_max_triplets_per_h* !E]	30
[*tangent_min_triplets_per_h* !E]	1
[*tangent_scale_difference_by* !E]	1
[*tangent_tiny* !E]	0.01
Miscellaneous	
[*apply_exp_scale* !E]	1
[*correct_for_atomic_scattering_factors* !E]	1
[*correct_for_temperature_effects* !E]	1
[*hkl_plane* $hkl]*...*	
[*randomize_initial_phases_by* !E]	Rand(-180,180)
[*scale_E* !E]	1

[*scale_F* !E]	1
[*scale_F000* !E]	0
[*scale_weak_reflections* !E]	
[*user_threshold* !E]	
[*verbose* #]	1
GUI Related	
[*add_to_cloud_N* !E [*add_to_cloud_when* !E]]	
[*pick_atoms_when* !E]	
[*view_cloud* !E]	1

20.1 ... Charge-flipping usage

CF works well on data at good resolution (<1Å resolution). For data at poor resolution or for diffi-
cult structures then inclusion of the tangent formula can facilitate solution and sharpen electron
densities, see example CF-1A7Y.INP. Powder diffraction data usually fall under the poor resolu-
tion/data quality category and as such additional symmetry restraints using *sym-*
metry_obey_0_to_1 can sharpen electron densities. Example CF-ALVO4.INP demonstrates the use
of the tangent formula on powder data.

The choice and amount of perturbation necessary for finding a solution are important considera-
tions. Not enough perturbation leads to the system being trapped within a local parameter space;
too much perturbation may lead to a solution not being found and in addition contrast in R-factors
prior to and at convergence are diminished leading to difficult to identify solutions. Many of the
examples in the CF directory uses the *Ramp* macro to gradually vary control parameters, here are
some examples:

```
fraction_density_to_flip = Ramp(0.85, 0.8, 100);
fraction_reflections_weak = Ramp(0.5, 0, 100);
flip_regime_2 = Ramp(1, 0, 200);
flip_regime_3 = Ramp(0.25, 0.5, 200);
symmetry_obey_0_to_1 = Ramp(0.5, 1, 100);
tangent_scale_difference_by = Ramp(0, 1, 100);
```

Choosing control *parameters* in this manner gradually decreases perturbation allowing for solu-
tions to be found and identified. This is similar-to a simulated annealing process where tempera-
tures start at high values and then progressively lowered.

20.1.1 ....Perturbations

Perturbations can be categorized as being of either phase, structure factor intensity or electron
density perturbations as shown in Table 20-2. There are two built in flipping regimes, *flip_re-*
gime_2 and *flip_regime_3*, and one user defined regime *flip_equation*. Only one can be used and
they all modify the electron density. In the absence of a flipping regime the following is used where
δ corresponds to the electron density threshold.

$$\rho = \begin{cases} -\rho & for \ \rho < \delta \\ \rho & for \ \rho \ \leq \delta \end{cases} \tag{20-1}$$

Using the tangent formula on either difficult structures or on data at poor resolution often leads to uranium atom solutions. Uranium atom solutions can be avoided by modifying the electron density using a flipping regime that dampens high electron densities or by using *pick_atoms*.

Using a largenumber of triplets per E_h value (a value for *tangent_max_triplets_per_h* greater than 100) reduces perturbation, increases occurrences of uranium atom solutions and increases the chances of finding a solution after an initial phase randomization. A large number of triplets would typically be used for poor resolution data; correspondingly a flipping regime that avoids uranium atom solutions should be chosen. Perturbations mostly increase randomness in the system with the exceptions of the tangent formula, *scale_density_below_threshold* and *histogram_match_scale_fwhm*.

20.1.2 .... The Ewald sphere, weak reflections and CF termination

By default, CF uses the minimum observed *d* spacing to define the Ewald sphere; alternatively, *min_d* can be used. The Ewald sphere can be increased using *extend_calculated_sphere_to*; this inserts missing reflections and gives them the status of "weak". Weak reflections are also inserted for missing reflections within the Ewald sphere. Weak reflection phases and structure factors can be modified using *scale_weak_reflections* and *add_to_phases_of_weak_reflections*.

Reflections that have zero intensities according to the space group are not included in CF; correspondingly the number of observed reflections removed are reported. Structure factor intensities within a family of reflections are determined by averaging the observed structure factors intensities. This averaging is also performed on calculated intensities each CF iteration for weak reflections.

Changing the space group is possible; changing the space group to a higher symmetry from that as implied in the input hkl file often makes sense. Changing the space group to a lower symmetry implies less symmetry and is useful for checking whether a significantly better R-factor is realized.

Typically, a fraction of observed reflections is given the status of "weak" using *fraction_reflections_weak*. When a solution is found and CF terminates, a *.FC file is saved; this file comprises structures factors that produced the best R-factor. A new CF process can be initiated with phase information saved in the *.FC file using the *Restart_CF* macro. To further complete the structure the new CF process may for example reduce perturbations in order to sharpen the electron density.

20.1.3 .... Powder data considerations

For powder data it is usually best to maximize the number of constraints due to poor data quality; it is also best to use *.A files as generated by a Pawley refinement and to then use *cf_in_A_matrix*. No weak observed reflections within the observed Ewald sphere should be assigned by setting *fraction_reflections_weak* to zero. Instead weak reflections can be included by extending the Ewald sphere with something like:

```
      extend_calculated_sphere_to 1
      add_to_phases_of_weak_reflections = 90 Ramp(1, 0, 100);
```

If the Ewald sphere is extended such that the weak reflections are many then some of these weak reflections could well be of high intensity. Subsequently offsetting high intensity weak reflections by a constant could lead to too much perturbation and thus the following may be preferential:

```
      extend_calculated_sphere_to 1
      add_to_phases_of_weak_reflections = Rand(-180,180) Ramp(1, 0, 100);
```

In a Pawley refinement the calculated intensities at the low *d*-spcaing edge of are often in error to a large extent; it is therefore best to remove these reflections using *delete_observed_reflections*, for example example:

```
      delete_observed_reflections = D_spacing < 1.134;
```

A typical first try INP file template for powders is as follows:

```
      macro Nr { 100 }
      charge_flipping
         cf_in_A_matrix PAWLEY_FILE.A
         space_group $
         a # b # c al # be # ga #
         delete_observed_reflections = D_spacing < #;
         extend_calculated_sphere_to #
         add_to_phases_of_weak_reflections = 90 Ramp(1, 0, Nr);
         flip_regime_2 = Ramp(1, 0, Nr);
         symmetry_obey_0_to_1 = Ramp(0.5, 1, Nr);
         Tangent(0.3, 30)
         min_grid_spacing 0.3
         load f_atom_type f_atom_quantity { ... }
```

20.1.3.1...... Powder data, the A matrix and the Tangent formula

In the case of charge-flipping from powder data TOPAS uses the diagonally normalized A-matrix *cf_in_A_matrix* (see example CF\CF-CIME.INP), which we will call **D,** from a Pawley refinement (see example CF\CF-CIME-PAWLEY.INP) to modify normalized structure factors $\mathbf{E}_h$ calculated during the charge-flipping process; this produces better results than using reflections output in a SHELX format (Whitfield & Coelho, 2016). Equation (20-2) shows how the structure factors are modified.

$$\mathbf{E}_{h,calc,modified} = \mathbf{E}_{h,calc} \sqrt{\frac{I_{obs,h,w}}{I_{calc,h,w}}} \tag{20-2}$$

where $\quad I_{calc,h,w} = \sum_k \mathrm{D}_{h,k}^2 \, I_{calc,k} \quad$ and $\quad I_{obs,h,w} = \sum_k \mathrm{D}_{h,k}^2 \, I_{obs,k}$

The subscripts *h* and *k* correspond to reflections *h* and *k* respectively and the summation in *k* is over all reflections. $I_{calc,k}$ and $I_{obs,k}$ corresponds to observed and calculated intensities. Equation (1) modifies the calculated intensities to include intensities from overlapping peaks. When there's no overlap $D_{i,i}$=1 and $D_{i,j}$=0 and the calculated intensities as well as $\mathbf{E}_h$ are not modified. When using

the direct-methods tangent formula within the charge-flipping process as described by Coelho (2007), the **D** matrix is also used to modify E_h values used in triple phase relationships as shown in equation (2).

$$E_{obs,h,modified} = E_{obs,h}q_h + E_{calc,h}(1 - q_h) \qquad (20\text{-}3)$$
$$\text{where} \quad q_h = \textstyle\sum_k \mathrm{D}_{h,k}^2$$

$E_{obs,h}$ and $E_{calc,h}$ corresponds to tangent formula E_h values calculated from the observed and calculated intensities respectively. $E_{calc,h}$ is typically not used in the tangent formula, however, intensities used for determining $E_{obs,h}$ can be grossly in error due to peak overlap. Equation (2) therefore influences triple phase relationships by weighing $E_{obs,h}$ by q_h; when there's no overlap q_h=1 resulting in no modification. When there's significant overlap then q_h is small and the influence of triple phase relationships using the h reflection is reduced. Equation (2) also includes a $(1-q_h)$ portion of $E_{calc,h}$ thus stating that when there's significant overlap the calculated E_h is to be more trustworthy than the observed E_h. Equation (20-3) corrects for errors in E_h when I_{obs} is similar-to I_{calc}; this assists in reducing the goodness of fit value thus enhancing the chances of solving the structure.

20.1.3.2...... The algorithm of Oszlányi & Süto (2005) and F000

Normalized structure factors enhance the chances of finding a solution (Oszlányi & Süto, 2006) and are realized by inclusion of *f_atom_type*'s and when *correct_for_temperature_effects* is non-zero. Example CF-1A7Y-GABOR.INP implements the algorithm of Oszlányi & Süto (2005) with normalized structure factors. In the original algorithm the amount of charge flipped is a function of the maximum electron density; this can be realized using:

```
user_threshold = 0.2 Get(max_density_at_cycle_iter_0);
```

Get(max_density_at_cycle_iter_0) is a different value at the start of each CF process as phases are chosen randomly. An alternative means of defining the threshold is:

```
fraction_density_to_flip 0.83
```

The CF process is sensitive to the threshold value. A value of 0.83 for *fraction_density_to_flip* is optimum for 1A7Y and produces a solution in ~1000 iterations. A solution is not found however at 0.75 or 0.85. To overcome this sensitivity the *fraction_density_to_flip* parameter could be ramped as a function of iteration from a high value to a low value, or,

```
fraction_density_to_flip = Ramp(0.85, 0.8, 100);
```

Implementation of such a ramp solves 1a7y in ~2000 iterations.

F000 is allowed floats when *scale_F000* is set to 1. In the Oszlányi & Süto (2005) algorithm a floating F000 produces the best results for some structures but not for others (see section 20.2.3). When the electron density is perturbed then a floating F000 often produces unfavourable oscillations in R-factors. In general, the electron density is best left unperturbed when *scale_F000* is non-zero. Example CF-1A7Y-GABOR.INP does not seem to solve at a lower resolution, try for example:

```
      delete_observed_reflections = D_spacing < 1.1;
```

On the other hand, when *scale_F000* is zero then electron density perturbations are possible; CF_1A7Y.INP solves 1A7Y at 1.1 Angstrom (i.e. include "*delete_observed_reflections* = *D_spacing* < 1.1"); CF_1A7Y.INP uses *flip_regime_2* and the tangent formula.

20.2 ... Charge-flipping Investigations / Tutorials

The effects of CF keywords can be investigated by inclusion/exclusion of keywords or by changing equations. This section lists some investigative examples and highlights the use of keywords necessary to solve examples found in the CF directory.

20.2.1 ....Preventing uranium atom solutions using pick_atoms

Example CF-1A7Y-OMIT.INP uses *pick_atoms* to modify the peaks of the highest intensity atoms, or,

```
      pick_atoms *      choose_to 5      omit = Rand(1, 1.1);
```

This example additionally uses the tangent formula and 1A7Y solves in ~100 iterations and with a large contrast in R-factors before and at converegnce. Another means to modify the peaks are:

```
      pick_atoms *      choose_to 5      insert = Rand(-.1, 1);
```

The *insert* case is slightly slower than the *omit* case as the 5 atoms are first omitted before insertion. Each case however solves 1A7Y in a similar number of iterations.

Example CF-1A7Y-NO-TANGENT.INP is similar but without the tangent formula, 1A7Y in this case solves in ~1000 iterations.

20.2.2 ....The tangent formula on powder data

In CF-ALVO4.INP comment out the Tangent line as follows:

```
      ' Tangent(.5, 50)
```

Run CF-ALVO4.INP and turn on Octahedra viewing in the OpenGL window. Visual inspection of picked atoms should show electron densities that are not recognizable as correct solutions.

Include the Tangent line and rerun; after a minute or two and at the bottom of the Ramps visual inspection of picked atoms should show a welldefined solution.

Thus, use of the tangent formula assists in solving CF-ALVO4.INP.

20.2.3 ....Pseudo symmetry – 441 atom oxide

CF works well on pseudo symmetric structures (Oszlányi *et al.*, 2006). Example CF-PN-02.INP is an oxide structure that contains 441 atoms in the asymmetric unit (Lister *et al.*, 2004); run CF to convergence. Pick atoms and turn on Octahedra viewing; all polyhedra should be well formed. Thus,

CF works extremely fast and trivializes the solving of such structures. The contents of the INP file is as follows:

```
charge_flipping
   cf_hkl_file 020pn.hkl
   space_group Pn
   a 24.1332  b 19.5793  c 25.1091  be 99.962
   fraction_reflections_weak 0.4
   symmetry_obey_0_to_1 0.3
   Tangent(0.25, 30)
   load f_atom_type f_atom_quantity {
      MO = 42 2;
      P  = (126 - 42) 2;
      O  = (441 - 126) 2;
   }
```

The tangent formula is used to assist *symmetry_obey_0_to_1* and to assist in finding the solution faster; it is not necessary however for this example. The Oszlányi & Süto (2005) algorithm can be used by replaceing *symmetry_obey_0_to_1* and the Tangent line with the following:

```
scale_F000 1
fraction_reflections_weak 0.4
add_to_phases_of_weak_reflections 90
user_threshold = 0.15 Get(max_density_at_cycle_iter_0);
```

Slow convergence is observed due to the use of F000. This is opposite to the case of 1a7y in CF-1A7Y-GABOR.INP where F000 is necessary. Setting *scale_F000* to zero greatly increases the rate of convergence.

20.2.4 ....Origin finding and symmetry_obey_0_to_1

When *symmetry_obey_0_to_1 is* defined origin finding is performed each iteration of charge flipping. Symmetry elements of the space group are used in finding an origin. On finding an origin the electron density is shifted to a position that best matches the symmetry of the space group. Additionally, a restraint is placed on the electron density pixels forcing symmetry to be obeyed.

Run CF-AE14.INP to convergence; notice the *P-1* symmetry. Remove *symmetry_obey_0_to_1* and run to convergence; the origin should now be arbitrary.

20.2.5 ....symmetry_obey_0_to_1 on poor resolution data

Run CF-AE5.INP until a solution is found; terminate CF, this saves the phase information to the file AE5.FC. Copy AE5.FC to AE5-SAVE.FC. Place the following lines into the file CF-AE5-POOR.INP:

```
set_initial_phases_to ae5-save.fc
randomize_initial_phases_by 0
```

This simply starts CF with optimum phase values. Also include the following line:

```
symmetry_obey_0_to_1 0.75
```

Run `CF-AE5-POOR.INP`; the atom positions after picking should visually produce the correct result. Comment out *symmetry_obey_0_to_1* and rerun CF-AE5-POOR.INP. R-factors should diverge and picked atoms should show a non-solution. Thus, *symmetry_obey_0_to_1* assists in solving CF-AE5-POOR.INP.

Include *symmetry_obey_0_to_1* and remove *set_initial_phases_to* and *randomize_initial_phases_by* and then rerun CF-AE5-POOR.INP. A solution should be obtained in a few minutes. Note that in this example the default flipping regime leads to regular occurrences of uranium atom solutions; this can be trivially ascertained by viewing the electron density. To reduce the occurrences of uranium atom solutions the following flipping regime is used:

```
flip_regime_3 0.5
```

20.2.6 ....Sharpening clouds - extend_calculated_sphere_to

Example `CF-AE9-POOR.INP` demonstrates the limit to which the present CF implementation can operate. Single crystal data is purposely chosen to isolate resolution effects and not intensity errors. The tangent formula is critical where without it the CF process is extremely perturbed and unstable. '*flip_regime_3* 0.5' is used due to occurrences of uranium atom solutions.

There are no ramps, instead the CF process is restarted when the R-factor fails to decrease for 100 consecutive iterations, or,

```
break_cycle_if_true = Get(iters_since_last_best) > 100;
randomize_phases_on_new_cycle_by = Rand(-180, 180);
```

Half of the observed reflections are considered weak and additionally missing reflections up to 1 Angstrom are included and considered weak using:

```
fraction_reflections_weak 0.5
extend_calculated_sphere_to 1
```

The intensities of weak reflections are left untouched and instead a Pi/2 phase shift is randomly applied to ~30% of weak reflections as follows:

```
add_to_phases_of_weak_reflections = If(Rand(0, 1) < .3, 90, 0);
```

A *symmetry_obey_0_to_1* of 0.7 is used not merely to find an origin but rather to prevent the electron density from straying.

Run `CF-AE9-POOR.INP` and a solution should be clearly recognizable after a few minutes. Change/remove keywords and rerun to view effects. Examples `CF-CIME-POOR.INP` and `CF-AE5-POOR.INP` are similar.

20.2.7 ....A difficult powder, CF-SUCROSE.INP

CF-SUCROSE.INP without *scale_density_below_threshold*=0 exhibits large oscillations in R-factors resulting in difficult to identify solutions; this can be prevented by increasing the amount of charge flipped and including *scale_density_below_threshold*=0, for example

```
    fraction_density_to_flip 0.83
    scale_density_below_threshold 0
```

When *scale_density_below_threshold=0* is used the percentage of charge that is less than the threshold before the application of *scale_density_below_threshold* is reported; the difference between this reported value and (1-*fraction_density_to_flip*) gives the flipped pixels that survived *scale_density_below_threshold*. At *fraction_density_to_flip* of 0.83 approximately 23% of pixels survives *scale_density_below_threshold*=0 which in effect means that only 23% of pixels are actually flipped out of the original 83%.

The following can be used to omit 30% of atoms:

```
    pick_atoms *
        activate = Cycle_Iter == 0;
        insert = If(Rand(0, 1) > 0.3, 10, 0);
```

Note that atoms are inserted at an intensity that is 10 times the average intensity. This increases the weight of inserted atoms relative to electron density noise. It also initially gives more weight to weak reflections.

Use of *scale_density_below_threshold* often results in CF requiring more interations to solution; a solution however is preferable to no solution.

20.2.8 ....Increasing contrast in R-factors

The act of flipping introduces an appreciable amount of unwanted high frequencies in the structure factors. This effect can be reduced by dampening high frequencies using *apply_exp_scale* which is ON by default. *apply_exp_scale changes* R-factors and not phases, directions taken by CF are unchanged.

Run CF-1A7Y.INP until convergence. The difference in R-factors before and at convergence should be ~0.39 (i.e. 0.81 and 0.42). Turn OFF *apply_exp_scale* by including the following line:

```
    apply_exp_scale 0
```

Rerun CF-1A7Y.INP until convergence. The difference in R-factors before and at convergence should now be ~0.29 (i.e. 0.81 and 0.52). Thus *apply_exp_scale* increases contrast in R-factors. Note that most of the increase seems to be realized from d-spacings less than 1 Angstrom.

20.3 ... Charge Flipping and neutron_data

neutron_data informs the charge flipping routine that neutron scattering lengths are to be used. It also results in the following default neutron flipping routine being used:

```
    flip_equation =
        If(And(Get(density)< Get(threshold),Get(density) > 0.4 Get(min_density)),
            -Get(density),
             Get(density)
        );
```

flip_neutron can be used to change the 0.4 value occurigng in the above equation, for example:

```
flip_neutron = 0.5;
```

The tangent formula is made less accurate due to the negative scattering of H atoms. However, if positive scattering lengths are dominant then the tangent formula can stabilize refinement. For example (see TEST_EXAMPLES\CF\NEUTRON-CIME\CF-NEUTRON.INP), try:

```
Tangent(0.3, 30) tangent_scale_difference_by = Ramp(1, 0, Nc);
```

20.4 ... Charge-flipping Examples

Table 20-3. Examples found in the CF directory. Number of atoms corresponds to the number of non-hydrogen atoms within the asymmetric unit.

Single crystal data	Num atoms in asymmetric unit	Space group
CF-1A7Y.INP CF-1A7Y-GABOR.INP CF-1A7Y-OMIT.INP CF-1A7Y-NO-TANGENT.INP	314	*P1*
CF-AE14.INP	43	*P-1*
CF-AE5.INP CF-AE5-POOR.INP	23	*C2/c*
CF-AE9.INP CF-AE9-POOR.INP	53	*P-1*
CF-GEBAA.INP	17	$P4_12_12$
CF-PN-02.INP	441	*Pn*
CF-YLIDM.INP	17	$P2_12_12_1$
Powder data		
CF-ALVO4.INP CF-ALVO4-PAWLEY.INP	18	*P-1*
CF-CIME-PAWLEY.INP CF-CIME.INP CF-CIME-HISTO.INP CF-CIME-POOR.INP CF-CIME-POOR-HISTO.INP	17	$P2_1/a$
CF-SUCROSE.INP CF-SUCROSE-PAWLEY.INP	23	$P2_1$

20.5 ... Keywords in detail

[*add_to_cloud_N* !E]

  [*add_to_cloud_when* !E]

  The current cloud is added to the GUI cloud creating a running average cloud for display pur-poses. *add_to_cloud_N* corresponds to the number of most recent clouds to include in the running average. *add_to_cloud_when* determines when the current cloud is to be included in the running average; here's an example:

```
add_to_cloud_N 10 add_to_cloud_when = Mod(Cycle_Iter, 2);
```

  Averaged clouds eliminate noise and is effective if the cloud remains stationery which is gen-erally the case. Note that *add_to_phases_of_weak_reflections* can produce translations of the cloud and should not be included when averaging clouds.

[*add_to_phases_of_weak_reflections* !E]

  Allows for modification to phases of weak reflections. For example, to add $\pi/2$ to the phases of weak reflections then the following could be used:

```
add_to_phases_of_weak_reflections 90
```

  When *add_to_phases_of_weak_reflections* is defined then the intensities of weak reflections are not set to zero; instead they are left untouched meaning that their intensities are set to the values as determined by the inverse Fourier transform. See also *scale_weak_reflections*.

[*apply_exp_scale* !E]

  Determines *a* and *b* each CF iteration such that the following is a minimum:

  R-factor = $\sum | a \, \mathrm{Exp}(b \, / \, D\_spacing\text{^}2) \, Fc - Fo |$

  where *Fc* and *Fo* are the calculated and observed moduli respectively. Use of *apply_exp_scale* corrects R-factors in case of an incorrect temperature factor correction as applied when nor-malizing structure factors. Use of *apply_exp_scale* typically increases the difference between R-factors prior to and at convergence. *apply_exp_scale* is used by default, setting it to zero prevents its use.

[*cf_hkl_file* $file]

  Defines the input hkl file.

[*cf_in_A_matrix* $file [*scale_Aij* !E]]

  Data input is from a file created using *out_A_matrix* from a previous Pawley refinement. The correlations in $file are used to partition intensities during each iteration of charge-flipping. This partitioning is applied to structure factors as used by CF and as used by the tangent for-mula. *scale_Aij* can be used to modify the A matrix off-diagonal coefficients, here are some examples:

```
scale_Aij = Get(Aij);
scale_Aij = Get(Aij)^2; ' The default
scale_Aij = 0;          ' Equivalent to using a Pawley generated hkl file
```

CF on powder data can also be initiated using standard hkl files.

[*break_cycle_if_true* !E]

Interrupts charge flipping to execute *randomize_phases_on_new_cycle_by. Cycle_Iter* is set to zero and *Cycle* is incremented.

[*correct_for_atomic_scattering_factors* !E]

Structure factors are normalized when *correct_for_atomic_scattering_factors* is non-zero and when *f_atom_type*'s are defined. By default structure factors are normalized.

[*correct_for_temperature_effects* !E]

Attempts to remove isotropic temperature effects from the structure factors. *correct_for_temperature_effects* is ON by default, setting it to zero will prevent the correction. Normalized structure factors are realized when *correct_for_temperature_effects* is ON and the unit cell contents is defined using *f_atom_type* and *f_atom_quantity*.

[*delete_observed_reflections* !E]

Reflections are deleted before entering CF according to *delete_observed_reflections*; it can be a function of *D_spacing*, for example:

```
delete_observed_reflections = D_spacing < 1.1;
```

Once deleted, observed reflections cannot be reinstated by changing *min_d*.

[*extend_calculated_sphere_to* !E]

Used to sharpen electron density clouds by filling in missing reflections; added reflections are given the status of "weak". *extend_calculated_sphere_to* can be used in conjunction with *scale_weak_reflections* and *add_to_phases_of_weak_reflections* to modify "weak" reflection magnitudes and phases respectively (see section 20.2.6); here's an example:

```
extend_calculated_sphere_to 1
add_to_phases_of_weak_reflections = If(Rand(0, 1) < .3, 90, 0);
```

[*f_atom_type* $type *f_atom_quantity* !E]...

Defines atom types and number of atoms within the unit cell; used by the tangent formula in determining E_h values and by the OpenGL display for picking atoms. For the tangent formula then realtive quantities are important.

[*find_origin* !E]

If defined and non-zero then origin finding is turned ON. *symmetry_obey_0_to_1* defines *find_origin* by default. *symmetry_obey_0_to_1* can be used without *find_origin* by defining and setting *find_origin* to zero.

[*flip_equation* !E]

Allwows for a user defined flip; here's an example:

```
flip_equation = If(Get(density)<Get(threshold),-Get(density), Get(density));
```

[*flip_regime_2* !E]

The elctron density is modified according to Eq. (20-1) and then further modified using:

$$\rho = \rho - Get(\text{flip\_regime\_2})\rho^3/\rho_{max}^2$$

flip_regime_2 is typically ramped from 1 to 0.

[*flip_regime_3* !E]

The elctron density is modified according to Eq. (20-1) and then further modified using:

$$\rho = \begin{cases} \rho, & for\, \rho < \delta \\ Min(\rho, \rho_{max}Get(\text{flip\_regime\_3})), & for\, \rho \geq \delta \end{cases}$$

A value of 0.5 for *flip_regime_3* introduces little perturbation whilst reducing the occurance of uranium atom solutions. It is recommended that *flip_regime_3* be used in cases where *flip_regime_2* produces uranium atom solutions. An additional perturbation, such as "*add_to_phases_of_weak_reflections*=90;" may be necessary.

[*fraction_density_to_flip* !E]

The amount of charge flipped is fractionally based. A value of 0.6, for example, sets the threshold δ such that the sign of the lowest 60% of charge is changed. Get(*threshold*) can be used to retrieve δ.

[*fraction_reflections_weak* !E]

Defines the fraction of observed reflections to flag as "weak". When *scale_weak_reflections*, *add_to_phases_of_weak_reflections* and *extend_calculated_sphere_to* are all not defined then intensities of weak reflections are set to zero effectively removing them from the charge flipping process. Otherwise intensities of weak reflections are not set to zero; instead they are left untouched prior to *scale_weak_reflections* and *add_to_phases_of_weak_reflections* and space group family averaging.

[*histogram_match_scale_fwhm* !E]
    [*hm_size_limit_in_fwhm* !E]
    [*hm_covalent_fwhm* !E]

An implementation of Histogram Matching ([b]Baerlocher *et al.*, 2007) where the distribution of pixels within the unit cell is restrained to one that matches Gaussian atoms with intensities corresponding to the atoms defined by *f_atom_type*'s. The Histogram matching operation is performed when *histogram_match_scale_fwhm* evaluates to a non-zero value. Subsequently the full width at half maximum (FWHM) of the Gaussians (obtained from the file ATOM_RA-DIUS.DEF) is scaled by *histogram_match_scale_fwhm. hm_size_limit_in_fwhm* corresponds to the extent to which the Gaussians are calculated in units of FWHM. Covalent radii is used if *hm_covalent_fwhm evaluates* to a non-zero value otherwise ionic radii is used. An example use is as follows:

```
histogram_match_scale_fwhm = If(Mod(Cycle_Iter, 3), 0, 1);
    hm_size_limit_in_fwhm 1
    hm_covalent_fwhm 1
```

Reported on is the fraction of pixels modified; values of 1 for both *histogram_match_scale_fwhm* and *hm_size_limit_in_fwhm* seem optimal where typically ~15 to 20% of pixels are modified. Use of histogram matching should produce R-factors at convergence that are equal to or than less R-factors produced when not using histogram matching. Histogram matching sharpens the electron density cloud for data at poor resolution (see examples CF-CIME-HISTO.INP and CF-CIME-POOR-HISTO.INP).

[*min_d* !E]

Determines in Å the resolution of observed reflections to work with; only observed reflections with a d-spacing above *min_d* are considered. *min_d* is evaluated each CF iteration. Get(*num_observed_reflections_above_d_min*) is updated when a change in *min_d* is detected. See also *extend_calculated_sphere_to* and *delete_observed_reflections*.

[*min_grid_spacing* !E]

If defined, then the grid spacing used is set to the smaller of *min_d*/2 and *min_grid_spacing*; useful for obtaining many grid points for graphical purposes.

[*neutron_data*]

Signals that the input data is of neutron type. Used in the picking of atoms and additionally E_h values are not corrected from any defined *f_atom_type* and *f_atom_quantity* keywords.

[*pick_atoms* $atoms]...
    [*activate* !E]
    [*choose_from* !E]
    [*choose_to* !E]
    [*choose_randomly* !E]
    [*omit* !E]
    [*displace* !E]
    [*insert* !E]

pick_atoms modifies the electron density based on picked atoms. $atom corresponds to the atom types to be operated on; it can contain the wild card character '*' and the negation

character '!', see section 17.20 for details. The operations of *pick_atoms* are invoked when *activate* evaluates to a non-zero value, for example,

```
pick_atoms "O C"
    activate = Mod(Cycle_Iter, 20) == 0;
```

The picking routine attempts to locate the atom types found in $atom based on the intensities of picked atoms and the scattering power of the atoms defined in *f_atom_type*. For example,

```
load f_atom_type f_atom_quantity { Ca 2 O 10 C 12 }
pick_atoms "O C"
```

Here two Ca atoms are first picked and then 10 O atoms and then 12 C atoms. The picked atoms operated on will be the O and C atoms with the Ca atoms ignored.

choose_from and *choose_to* can be used to limit the number of atoms operated on. Note, that picked atoms within *pick_atoms* are sorted in decreasing intensity order. For example, to not operate on the first thee O atoms and the last 2 C atoms then the following could be used:

```
choose_from 4
choose_to 20
```

choose_randomly further reduces the atoms operated on and is executed after *choose_from* and *choose_to*.

omit removes operated-on-atoms from the electron density. Atoms can be partially removed by setting *omit* to values less than 1. Values greater than 1 can also be used, the effect is to change the sign of the electron density. *omit* operating on a few of the highest intensity atoms is an extremely effective means of preventing the occurance of uranium atom solutions, see CF-1A7Y-OMIT.INP; for example:

```
pick_atoms *
    choose_to 5
    omit = Rand(1, 1.1);
```

Omitting atoms randomly is a technique referred to as "random omit maps" in ShelXD, (Schneider and Sheldrick, 2002).

insert inserts operated on atoms; a value of 1 inserts the atoms with an intensity that is equal to the average of the picked atoms. Values of less than 1 decreases the intensity of the inserted atoms. When *insert* is defined then *omit* is internally defined if it does not already exist. Thus, atoms are removed before insertion by default.

displace displaces in Å atom positions from their picked positions; it is evaluated before *insert*. For example, to randomly displace atoms by 0.3 Å then the following could be used:

```
displace = Rand(0.4, 0.6);
insert 1
```

There can be more than one occurance of *pick_atoms*, for example to limit uranium atom solutions then the following can be used:

```
pick_atoms *
    choose_to 5
    insert = Rand(-.1, 1);
```

To further randomly remove ~33% of atoms then the following could additionally be used:

```
break_cycle_if_true = Get(iters_since_last_best) > 10;
pick_atoms *
    activate = Cycle_Iter == 0;
    insert = If(Rand(0, 1) > 0.33, 10, 0);
```

Note that in this example atoms are inserted at ten times the average picked intensity; this simply gives more weight to picked atoms relative to electron density noise. Additionaly weak reflections are also given more weighting.

[*pick_atoms_when* !E]

Atoms are picked in the OpenGL display when *pick_atoms_when* evaluates to a non-zero value; here's an example:

```
pick_atoms_when = Mod(Cycle_Iter + 1, 10) == 0;
```

Note that picking can be manually initiated from the Cloud dialog of the OpenGL display. A text description of picked atoms can be obtained by opening the "Temporary output" text window of the OpenGL window.

[*randomize_initial_phases_by* !E]

Initializes phases. To start a process with already saved phase information then the following could be used:

```
set_initial_phases_to aleady_saved.fc
randomize_initial_phases_by 0 ' this has a default of 0
```

[*randomize_phases_on_new_cycle_by* !E]

```
randomize_phases_on_new_cycle_by = Rand(-180, 180); ' an example
```

[*scale_density_below_threshold* !E]

Electron density pixels that are less than the threshold value are scaled by *scale_density_below_threshold*. Values for *scale_density_below_threshold* thata re less than 1 tends to sharpen the electron density and to reduce large oscillations in R-factors; the latter occurs for bad data, see example CF-SUCROSE.INP. A value of zero for *scale_density_below_threshold* results in "low density elimination" simlar to that of Shiono & Woolfson (1992).

[*scale_E* !E]

Normalized structure factors (E_h values) are a function of *correct_for_temperature_effects* and unit cell contents. *scale_E* allows for an additional scaling of E_h values.

[*scale_F* !E]

CF works with normalized structure factors by default. *scale_F* is an additional scaling of structure factors. The defualt *scale_F* broadens electron density peaks to avoid pixilation effects and is given by:

```
scale_F = Exp(-0.2 Get(d_squared_inverse));
```

[*scale_F000* !E]

Scale should be set to 1 for compliance with the algorithm of Oszlányi & Süto (2004). When *scale_F000* is non_zero then modifcations to the electron density produces unfavourable effects.

[*scale_weak_reflections* !E]

By default, weak reflection structure factors are set to zero; however, when either *scale_weak_reflections* or *add_to_phases_of_weak_reflections* is defined then weak reflections structure factors are instead modified accordingly, for example:

```
scale_weak_reflections = Rand(-0.2, 0.4);
```

scale_weak_reflections or *add_to_phases_of_weak_reflections* can be a function of *D_spacing*.

[*set_initial_phases_to* $file]
    [*modify_initial_phases* !E]

Sets initial phases to those appearing in $file. Typically, $file corresponds to a *.FC file saved in a previous charge-flipping process. *modify_initial_phases* is executed each iteration of CF; it can be used to restrain the phases of $file. For example,

```
modify_initial_phases = Get(initial_phase) + Min(Abs(Get(phase_difference)),45);
```

where phase_difference corresponds to the difference between the current phase and the initial phase; it has a value between ±90º. *modify_initial_phases* can be used to constrain phases to those as determined by high resolution transmission electron microscopy HRTEM ([a]Baerlocher *et al.*, 2007).

[*space_group* $]

If defined, then the *cf_hkl_file* is assumed to comprise merged hkls corresponding to the defined space group; otherwise the *cf_hkl_file* is assumed to be of space group type *P1*.

[*symmetry_obey_0_to_1* !E]

If a space group is defined then symmetry is adhered to according to *symmetry_obey_0_to_1*. *symmetry_obey_0_to_1* can be throught of as a real space electron density restraint; its value should range between 0 and 1. If 1 then symmetry is obeyed 100%; if 0 then for a particular set of equivalent grid points, as determined by the equivalent positions of the space group, an average density ρ_{avg} is obtained. The electron densities on the grid points are then adjusted as follows:

$$\rho_{new} = \rho(1 - symmetry\_obey\_0\_to\_1) + symmetry\_obey\_0\_to\_1\,\rho_{avg}$$

The text output 'symmetry error' as displayed when *symmetry_obey_0_to_1* is used is defined as follows:

$$'symmetry\ error' = \frac{\sum|\rho - \rho_{avg}|}{\sum|\rho|}$$

where the summation is over all electron density grid points. *symmetry_obey_0_to_1* defines *find_origin* by default. *find_origin* is applied before *symmetry_obey_0_to_1*. *find_origin* shifts the electron density such that an approximate error in 'symmetry error' is minimized; thus *find_origin* assists in the *symmetry_obey_0_to_1* restraint.

[*tangent_num_h_read* !E]
   [*tangent_num_k_read* !E]
   [*tangent_num_h_keep* !E]
   [*tangent_max_triplets_per_h* !E]
   [*tangent_min_triplets_per_h* !E]
   [*tangent_scale_difference_by* !E]

tangent_num_h_read and *tangent_num_k_read* defines the number of highest h and highest k reflections to read in determining triplets. *tangent_num_h_keep* defines the number of highest h reflections to include for tangent formula updating. *tangent_max_triplets_per_h* and *tangent_min_triplets_per_h* defines the maximum and minium number of triplets per reflection h. Reflections with less than *tangent_min_triplets_per_h* are not included for tangent formula updating. *tangent_scale_difference_by* corresponds to S in the following:

$$\emptyset_{h,new} = \emptyset_{h,cf} + S\,\alpha_h\big(\emptyset_{h,tf} - \emptyset_{h,cf}\big)$$

$$tan\big(\emptyset_{h,tf}\big) = T_h/B_h$$

$$T_h = \sum_k E_h E_k E_{h-k} sin(\emptyset_k - \emptyset_{h-k})$$

$$B_h = \sum_k E_h E_k E_{h-k} cos(\emptyset_k + \emptyset_{h-k})$$

$$\alpha_h = M_h/M_{h,max}\ ,\quad M_h = \sqrt{T_h^2 + B_h^2}$$

[*user_threshold* !E]

By default, Get(*threshold*) is determined using *fraction_density_to_flip*. When defined then *user_threshold* overrides *fraction_density_to_flip*. Electron density pixels are normalized to have a maximum value of 1, thus typical values for *user_threshold* range between 0 and 0.1.

[*use_Fc*]

Sets initial phases to those saved in a previous *.FC file. The FC file used corresponds to the same name as the data file, defined using *cf_hkl_file* or *cf_in_A_matrix*, but with a FC extension. *use_Fc* is similar-to *set_initial_phases_to* except that the file is implied.

[*verbose* #]

A value of 1 outputs text in a verbose manner. A value of 0 outputs text only when a R-factor less that a previous value is encountered within a particular Cycle.

[*view_cloud* !E]

Informs a detected GUI to display the electron density. Here are some examples:

```
view_cloud 1 ' Update cloud every charge-flipping iteration
view_cloud = Mod(Cycle_Iter, 10) == 0;
```

21. INDEXING

The following algorithm is based on the iterative method of Coelho (2003). Unlike *lp_serach* it requires the extraction of d-spacings. The INDEXING directory contains example INP files, example usage is as follows:

```
index_zero_error
try_space_groups "2 75"
load index_d {
   8.912
   7.126
   4.296
   ...
}
```

Individual space groups can be tried or for simplicity all Bravais lattices can be tried using standard macros as follows:

```
Bravais_Cubic_sgs
Bravais_Trigonal_Hexagonal_sgs
Bravais_Tetragonal_sgs
Bravais_Orthorhombic_sgs
Bravais_Monoclinic_sgs
Bravais_Triclinic_sgs
```

To try all unique extinction subgroup space-groups, a more exhaustive approach, then the following macros can be used:

```
Unique_Cubic_sgs
Unique_Trigonal_Hexagonal_sgs
Unique_Tetragonal_sgs
Unique_Orthorhombic_sgs
Unique_Monoclinic_sgs
Unique_Triclinic_sgs
```

On termination of Indexing a *.NDX file is created, with a name corresponding to the name of the INP file, and placed in the same directory as the INP file. The *.NDX file contains solutions found as well as a detailed summary of the best 20 solutions. Here's an example of an NDX file:

```
' Indexing method - Alan Coelho (2003), J. Appl. Cryst. 36, 86-95
' Time: 2.015 seconds
    'Sg      Status UNI     Vol      Gof     Zero      Lps ...
Indexing_Solutions_With_Zero_Error_2 {
   0) P42/nmc    3   0    1187.321    38.82    0.0000    11.1924  ...
   1) P42/nmc    3   0    1187.057    38.64    0.0000    11.1896  ...
   2) P42/nmc    3   0    1187.458    38.61    0.0000    11.1914  ...
   ...
}
/*
  ==================================================================
   0) P-1         0     985.652    30.80    0.0111     7.0877  ...

   h   k   l      dc       do     do-dc     2Thc     2Tho   2Tho-2Thc
   0   0   1   15.857   15.830   -0.027    5.569    5.578     0.009
```

```
     0   1   0    8.765    8.750   -0.015   10.084   10.101    0.017
      ...
    */
```

21.1 ... Figure of merit

The figure of merit M used in indexing is as follows:

$$M = \left[(1 + N_{uni}) d_{o,min}^2 \left(\frac{N_c}{N_o} \right) \sum_i |d_{o,i}^2 - d_{c,i}^2| Q_i \right]^{-1} \tag{21-1}$$

$$\text{where} \quad Q_i = w_i N_o / \Sigma_j \, w_j$$

Where d_o and d_c are the observed and calculated d-spacings, N_o and N_c the number of observed and calculated lines used, N_{uni} the number of unindexed lines and the summations are over the used observed indexing lines. Q_i is a weighting that assists in the determination of extinction subgroups where w_i could for example be the inverse of the error in the peak positions from a Pawley refinement (see INDEXING\MGIR\INDEX.INP). *index_I* correspond to w_i. The formulation of Q_i is such that with or without Q_i the figure of merit M is of the same order of magnitude. The reciprocal-space lattice relationship solved during the indexing process (Coelho, 2000) includes Q as follows:

$$\left[X_{hh} h^2 + X_{kk} k^2 + X_{ll} l^2 + X_{hk} h \, k + X_{hl} h \, l + X_{kl} k \, l + \frac{4 \, \pi Z_e}{360 \lambda^2} sin(2\theta) \right] W_{hkl} = \frac{W_{hkl}}{d_o^2} \tag{21-2}$$

$$\text{where} \quad W_{hkl} = Q_{hkl} d_o^m |\Delta 2\theta_{hkl}|$$

21.2 ... Extinction subgroup determination

At the end of an indexing run further indexing runs are internally performed across extinction subgroups (see section 21.8) to determine the most likely subgroup. These internal runs are seeded with already determined lattice parameters and in most cases the correct extinction subgroup is obtained without the need for Q_i in Eq. (21-1). Extinction subgroups can be explicitly searched using the macros defined TOPAS.INC, see for example *Unique_Orthorhombic_sgs*.

21.3 ... Reprocessing solutions - DET files

Details of solutions can be obtained at a later stage by including solution lines, found in the NDX file, in the INP file. For example, supposing details of solutions 50 and 51 were sought then the following (see example INDEXING\EX10.INP) could be used:

```
    index_lam 1.540596
    index_zero_error
    try_space_groups 2
    Indexing_Solutions_With_Zero_Error_2 {
      50) P-1      1   0    2064.788    9.74   0.0000   ...
      51) P-1      3   0    3128.349    9.61   0.0115   ...
    }
    load index_d {
       15.83 good
        8.75
        7.91
        ...
    }
```

After running this INP file, a *.DET file is created containing details of the supplied solutions.

21.4 ... Keywords and data structures

Tindexing
  [*index_lam* !E1.540596]
  [*index_min_lp* !E2] [*index_max_lp* !E]
  [*index_max_Nc_on_No* !E5]
  [*index_max_number_of_solutions* #3000]
  [*index_max_th2_error* !E0.05]
  [*index_max_zero_error* #0.2]
  [*index_th2* !E|*index_d* !E]...
      [*index_l* E1[*good*]]
  [*index_x0* !E]
  [*index_zero_error*]
  [*no_extinction_subgroup_search*]
  [*seed*[#]]
  [*try_space_groups* $]...
      [*x_angle_scaler* #0.1]
      [*x_scaler* #]

Values for most keywords are automatically determined or have defaults (appearing as numbers to the right) adequate for difficult indexing problems. In the following example from UPPW (service provided by Armel Le Bail to the SDPD mailing list at http://sdpd.univ-lemans.fr/uppw/), only a few keywords are necessary. Also note the use of *dummy*; this allows for the exclusion of 2θ and I values without having to edit the columns of data.

```
    seed
    index_lam  0.79776
    index_zero_error
    index_max_Nc_on_No 6
    try_space_groups 3
    load index_th2 dummy dummy index_I dummy  {
       ' d (A)  2Theta      Height        Area      FWHM
       1.724  26.50645     2758.3     23303.7      0.0450
       2.646  17.27733   150393.8    747063.6      0.0250
       3.235  14.13204    98668.8    493153.7      0.0250
       3.417  13.37776    11102.6     53185.0      0.0250
       5.190   8.80955      782.7      3910.9      0.0250
       ...
    }
```

21.5 ... Keywords in detail

[*index_lam* !E1.540596]

Defines the wavelength in Å.

[*index_min_lp* !E2.5] [*index_max_lp* !E]

Defines the minimum and maximum allowed lattice parameters. The maximum is typically automatically determined.

[*index_max_Nc_on_No* !E5]

Determines the maximum ratio of the number of calculated to observed lines. The value of 6 allows for up to 83% of missing lines.

[*index_max_number_of_solutions* #1000]

The number of best solutions to keep.

[*index_max_th2_error* !E0.05]

Used for determining impurity lines (un-indexed lines UNI in *.NDX). Large values, 1 for example, forces the consideration of more observed input lines. For example, if it is known that there are none or maybe just one impurity line then a large value for *index_max_th2_error* will speed up the indexing procedure.

[*index_max_zero_error* !E0.2]

Excludes solutions with zero errors greater than *index_max_zero_error*.

[*index_th2* !E | *index_d* !E]...
[*index_I* E1[*good*]]

index_th2 or *index_d* defines a reflection entry in 2θ degrees or d-spacing in Å. *index_I* is typically set to the area under the peak; it is used to weight the reflection. *good* signals that the corresponding d-spacing is not an impurity line. A single use of *good* on a large d-spacing

decreases the number of possible solutions and hence speeds up the indexing process (see example INDEXING\EX10.INP).

[*index_x0* !E]

Defines X_{hh} in the reciprocal lattice equation of (21-1). In a triclinic lattice the largest d-spacing can probably be indexed as 100 or 200 etc. Thus

```
index_x0 = 1/(dmax)^2;
```

speeds up the indexing process (if, in this case, the first line can be indexed as 100) and additionally the chances of finding the correct solution is enhanced, see EX13.INP. Note, if the data is in 2Th degrees then the following can be used:

```
index_x0 = (2 Sin(2Thmin Pi/360) / wavelength))^2;
```

The two macros *Index_x0_from_d* and *Index_x0_from_th2* simplify the use of *index_x0*.

[*index_zero_error*]

Includes a zero error.

[*no_extinction_subgroup_search*]

By default Extinction subgroup determination is performed at the end of an indexing run; this can be negated by defining *no_extinction_subgroup_search*.

[*seed* [#]]

Seeds the random number generator.

[*try_space_groups* $]...
　　[*x_angle_scaler* #0.1]
　　[*x_scaler* #]

Defines the space groups to be searched. The macros *Bravais_Cubic_sgs* etc... (see TOPAS.INC) defines lowest symmetry Bravais space groups. It is typically sufficient to use only these. Higher symmetry space groups for the Bravais lattices corresponding to the 10 best solutions is automatically searched at the end of an indexing run. Here are some examples of using *try_space_groups*.

Search	Use
Primitive monoclinic	`try_space_groups 3`
Monoclinic Bravais lattices of lowest symmetry	`Bravais_Monoclinic_sgs`
C-centered monoclinic of lowest symmetry	`try_space_groups 5`
All orthorhombic space groups individually	`Unique_Orthorhombic_sgs`

x_scaler is a scaling factor used for determining the number of steps to search in parameter space. *x_scaler* needs to be less than 1. Increasing *x_scaler* searches parameter space in finer detail. Default values are as follows:

Cubic	0.99	Orthorhombic	0.89
Hexagonal/Trigonal	0.95	Monoclinic	0.85
Tetragonal	0.95	Triclinic	0.72

x_angle_scaler is a scaling factor for determining the number of angular steps for monoclinic and triclinic space groups. Small values, 0.05 for example, increases the number of angular steps. The default value of 0.1 is usually adequate.

21.6 ... Identifying dominant zones

Here are two example output lines from an NDX file.

```
0) P42/nmc 3 0 1187.124   38.82 0.000 11.1904 11.1904 9.4799 90.00 90.00 90.00 ' ===  24 19
6) P-421c  3 0 1187.124   35.67 0.000 11.1904 11.1904 9.4799 90.00 90.00 90.00 ' ===  24 19
```

- The 1st column corresponds to the rank of the solution.
- The 2nd corresponds to the space group.
- The 3rd corresponds to the Status of the solution as follows:

  Status 1:   Weighting applied as defined in Coelho (2003).

  Status 2:   Zero error attempt applied.

  Status 3:   Zero error attempt successful and impurity lines removal successful.

  Status 4:   Impurity line(s) removed.

- The 4th column corresponds to the number of un-indexed lines.
- The 5th column corresponds to the volume of the lattice.
- The 6th corresponds to the goodness of fit value.
- The 7th corresponds to the zero error if *index_zero_error* is included.
- Columns 8 to 13 contains the lattice parameters.

The last two columns, let call them column Q_1 and Q_2, contain the number of non-zero $(h^2 + k^2 + h\,k)$ and l^2 values, respectively, used in the indexed lines. Q_1 and Q_2 represent the hkl coefficient for X0 and X1 respectively for Trigonal/Hexagonal systems. When Q_1=-999 or Q_2=-999 then the corresponding lattice parameters are not represented. This facility is useful for identifying dominant zones. For example, if the smallest lattice parameter is 3Å and the smallest d-spacings is 4Å then it is impossible to determine the small lattice parameter. In such cases values of –999 will be obtained. The following table gives the hkl coefficients corresponding to the Xnn reciprocal lattice parameters for the 7 crystal systems.

	X0	X1	X2	X3	X4	X5
Cubic	$h^2+k^2+l^2$					
Hexagonal, Trigonal	$h^2+k^2+h\,k$	l^2				

Tetragonal	h²+k²	l²				
Orhtorhombic	h²	k²	l²			
Monoclinic	h²	k²	l²	h l		
Triclinic	h²	k²	l²	h k	h l	k l

21.7 … *** Probable causes of Failure ***

The most probable cause of failure is the inclusion of too many d-spacings. If it is assumed that the smallest lattice parameter is greater than 3Å then it is problematic to include d-spacings with values less than about 2.5Å when there are already 30 to 40 reflections with d values greater than 2.5Å. Some of the problems caused by very low d-spacings are:

- The number of calculated lines increases dramatically and thus *index_max_Nc_on_No* will need to be increased.
- The low d-spacings are probably inaccurate due to peak overlap.

A situation where it is necessary to include low d-spacings is when there are only a few d-spacings available as in higher symmetry lattices.

21.8 … Space groups with identical absences – Extinction subgroups

The following table lists spaces groups than have identical hkls. Typically, an indexing run will identify one space-group from the extinction group.

Space group numbers with identical hkls	Space group symbols with identical hkls
Triclinic	
1 2	P1 P-1
Monoclinic	
9 15	Cc C2/c
5 8 12	C2 Cm C2/m
14	P21/c
7 13	Pc P2/c
4 11	P21 P21/m
3 6 10	P2 Pm P2/m
Orthorhombic	
70	Fddd
43	Fdd2
22 42 69	F222 Fmm2 Fmmm
68	Ccca
73	Ibca
37 66	Ccc2 Cccm
45 72	Iba2 Ibam
41 64	Aba2 Cmca

46 74	Ima2 Imma
36 40 63	Cmc21 Ama2 Cmcm
39 67	Abm2 Cmma
20	C2221
23 24 44 71	I222 I212121 Imm2 Immm
21 35 38 65	C222 Cmm2 Amm2 Cmmm
52	Pnna
56	Pccn
60	Pbcn
61	Pbca
48	Pnnn
54	Pcca
50	Pban
33 62	Pna21 Pnma
34 58	Pnn2 Pnnm
32 55	Pba2 Pbam
30 53	Pnc2 Pmna
29 57	Pca21 Pbcm
27 49	Pcc2 Pccm
31 59	Pmn21 Pmmn
26 28 51	Pmc21 Pma2 Pmma
19	P212121
18	P21212
17	P2221
16 25 47	P222 Pmm2 Pmmm

Tetragonal

142	I41/acd
110	I41cd
141	I41/amd
109 122	I41md I-42d
108 120 140	I4cm I-4c2 I4/mcm
88	I41/a
80 98	I41 I4122
79 82 87 97 107 119 121 139	I4 I-4 I4/m I422 I4mm I-4m2 I-42m I4/mmm
130	P4/ncc
126	P4/nnc
133	P42/nbc
103 124	P4cc P 4/mcc
104 128	P4nc P 4/mnc
106 135	P42bc P 42/mbc
137	P42/nmc
138	P42/ncm
134	P42/nnm
125	P4/nbm
114	P-421c
105 112 131	P42mc P-42c P42/mmc

102 118 136	P42nm P-4n2 P42/mnm
101 116 132	P42cm P-4c2 P42/mcm
100 117 127	P4bm P-4b2 P4/mbm
86	P42/n
85 129	P4/n P4/nmm
92 96	P41212 P43212
94	P42212
76 78 91 95	P41 P43 P4122 P4322
77 84 93	P42 P 42/m P4222
90 113	P4212 P-421m
75 81 83 89 99 111 115 123	P4 P-4 P4/m P422 P4mm P-42m P-4m2 P4/mmm

Trigonal & Hexagonal

161 167	R3c R-3c
146 148 155 160 166	R3 R-3 R32 R3m R-3m
184 192	P6cc P6/mcc
159 163 186 190 194	P31c P-31c P63mc P-62c P63/mmc
158 165 185 188 193	P3c1 P-3c1 P63cm P-6c2 P63/mcm
169 170 178 179	P61 P65 P6122 P6522
144 145 151 152 153 154 171 172 180 181	P31 P32 P3112 P3121 P3212 P3221 P62 P64 P6222 P6422
173 176 182	P63 P63/m P6322
143 147 149 150 156 157 162 164 168 174 175 177 183 187 189 191	P3 P-3 P312 P321 P3m1 P31m P-31m P-3m1 P6 P-6 P6/m P622 P6mm P-6m2 P-62m P6/mmm

Cubic

228	Fd-3c
219 226	F-43c Fm-3c
203 227	Fd-3 Fd-3m
210	F4132
196 202 209 216 225	F23 Fm-3 F432 F-43m Fm-3m
230	Ia-3d
220	I-43d
206	Ia-3
214	I4132
197 199 204 211 217 229	I23 I213 Im-3 I432 I-43m Im-3m
222	Pn-3n
218 223	P-43n Pm-3n
201 224	Pn-3 Pn-3m
205	Pa-3
212 213	P4332 P4132
198 208	P213 P4232
195 200 207 215 221	P23 Pm-3 P432 P-43m Pm-3m

21.9 ... Indexing Equations - Background

a, **b** and **c** lattice vectors can be converted to Cartesian coordinates with **a** collinear with the Cartesian *x* axis and **b** coplanar with the Cartesian *x–y* plane as follows:

$$\mathbf{a} = a_x \, \mathbf{i} \qquad\qquad \mathbf{b} = b_x \, \mathbf{i} + b_y \, \mathbf{j} \qquad\qquad \mathbf{c} = c_x \, \mathbf{i} + c_y \, \mathbf{j} + c_z \, \mathbf{k} \qquad\qquad (21\text{-}3)$$

where

$$a_x = a$$

$$b_x = b \cos(\gamma), \quad b_y = b \sin(\gamma)$$

$$c_x = c \cos(\beta), \quad c_y = c \, (\cos(\alpha) - \cos(\beta) \cos(\gamma)) \, / \sin(\gamma), \quad c_z{}^2 = c^2 - (c_x)^2 - (c_y)^2$$

a, *b*, *c* are the lattice parameters and α, β, γ the lattice angles. The reciprocal lattice vectors **A**, **B**, and **C** calculated from the lattice vectors of Eq. (21-3) become:

$$\mathbf{A} = A_x \, \mathbf{i} + A_y \, \mathbf{j} + A_z \, \mathbf{k} \qquad\qquad \mathbf{B} = B_y \, \mathbf{j} + B_z \, \mathbf{k} \qquad\qquad \mathbf{C} = C_z \, \mathbf{k}$$

The equation relating d-spacing d_{hkl} to hkl in terms of the reciprocal lattice parameters is:

$$X_{hh} h^2 + X_{kk} k^2 + X_{ll} l^2 + X_{hk} h \, k + X_{hl} h \, l + X_{kl} k \, l = 1/d_{hkl}^2 \qquad\qquad (21\text{-}4)$$

where

$$X_{hh} = A_x^2 + A_y^2 + A_z^2 \qquad\qquad X_{hk} = 2A_y B_y + 2A_z B_z$$

$$X_{kk} = B_y^2 + B_z^2 \qquad\qquad\qquad X_{hl} = 2A_z C_z$$

$$X_{ll} = C_z^2 \qquad\qquad\qquad\qquad X_{kl} = 2B_z C_z$$

22. GUI FUNCTIONALITY

1.1 ...... TOPAS is DPI aware

Monitors with a high number of Dots Per Inch (DPI), often display text that are too small. Windows can scale fonts using Windows font scaling to enlarge text. This scaling is carried through to TOPAS where fonts and bitmaps scale to the required size. Additionally, a thicker-text option ("Segoe UI Semibold") can be enabled if the TOPAS text appears too thin. The option is saved for subsequent TOPAS loads and is enabled/disabled from the View menu:
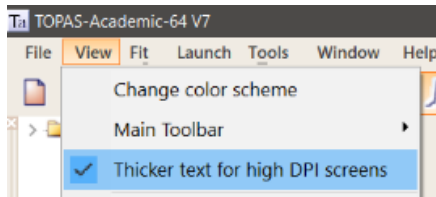
After applying Windows scaling, TOPAS needs a Sign-Out and a Sign-In to display all text correctly scaled. Also, for TOPAS to use its DPI capabilities, the properties of the executable needs to be set to "Application", i.e.
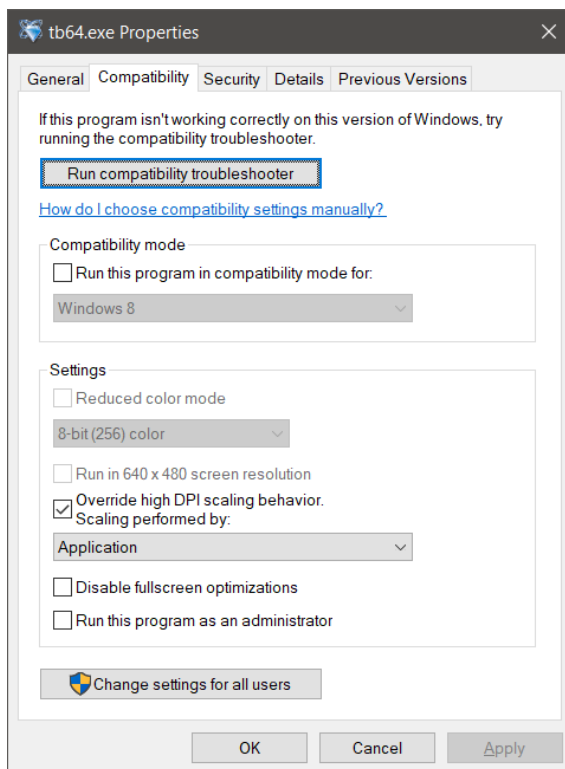
1.2 ...... Antialiasing and OpenGL

Enable Antialiasing on your graphics card to display smooth lines in OpenGL; this affects all OpenGL displays. Depending on the graphics card Antialiasing can also be enabled on a program specific manner.

1.3 ...... Displaying a phase with and without background

Phases can be plotted with or without background by cycling through the three states of the phase-display icon.

1.4 ...... How atoms are displayed in OpenGL

Atoms colours and radii are defined in the files ATOM_COLORS.DEF and ATOM_RADIUS.DEF respectively. A site defined as:

```
site S1 occ Al+3 1 beq 1
```

will be displayed as a Sulphur atom. If the Site Name, minus the numbers, is not found in ATOM_COLOURS.DEF then the atom type defined at the first site occupancy is used. Thus, a site defined as:

```
site _S1 occ Al+3 1 beq 1
```

will be displayed as an Aluminium atom.

1.5 ...... x_calculation_step deleted when constant x-axis step size detected

*.XY and *.XYE data files are converted to a constant x-axis step size when a constant step size is detected. When this occurs Version 7 removes the "Calc.Step" item from the GUI menus for the corresponding data file. A small calculation step size can still be used by increasing "Conv. Steps". PRO files containing an *x_calculation_step* will still show an entry of *x_calculation_step*.

1.6 ...... hide_peak_sticks

A GUI option that toggles the display of peak sticks in the scan window; the option can be found at the Peaks phase level as follows:
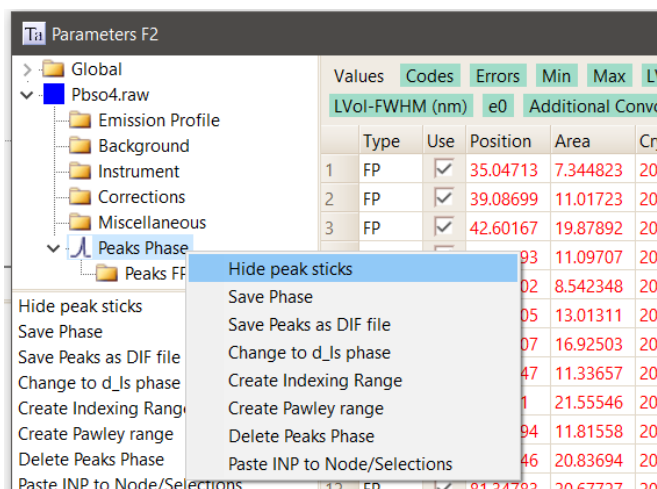
22.1 ... TOF x-axis can be displayed as d-spacing, Q or tof

The x-axis of TOF data can be displayed as either tof, d-spacing or Q by cycling the x-axis button.

22.1.1 ....Surface plots – 2D with offsets

This icon displays scans offset from one another, for example (see files in the directory TEST_EXAMPLES\3D\):
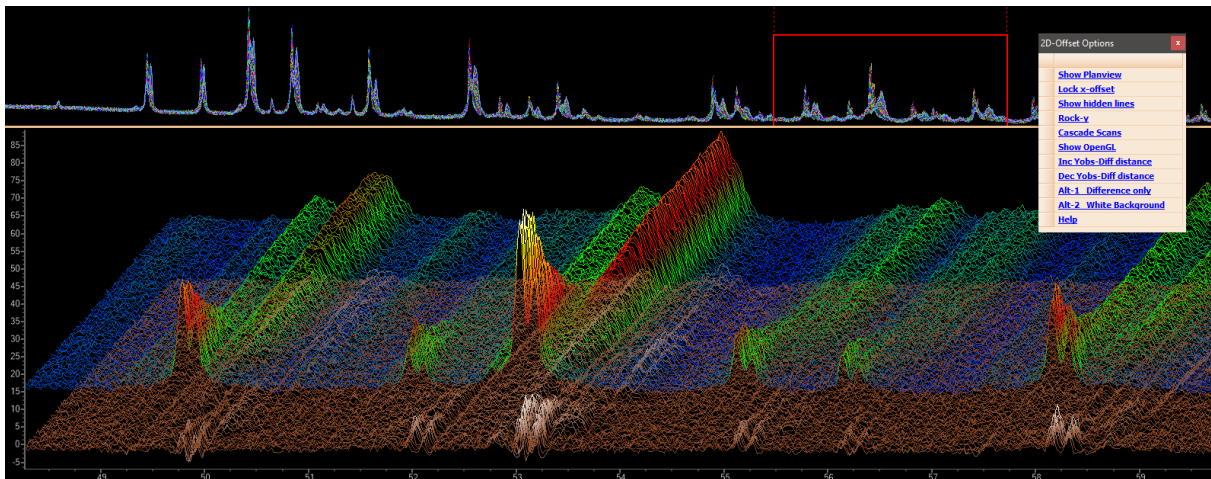
The Quickzoom window is operational in all 2D-offset plots.
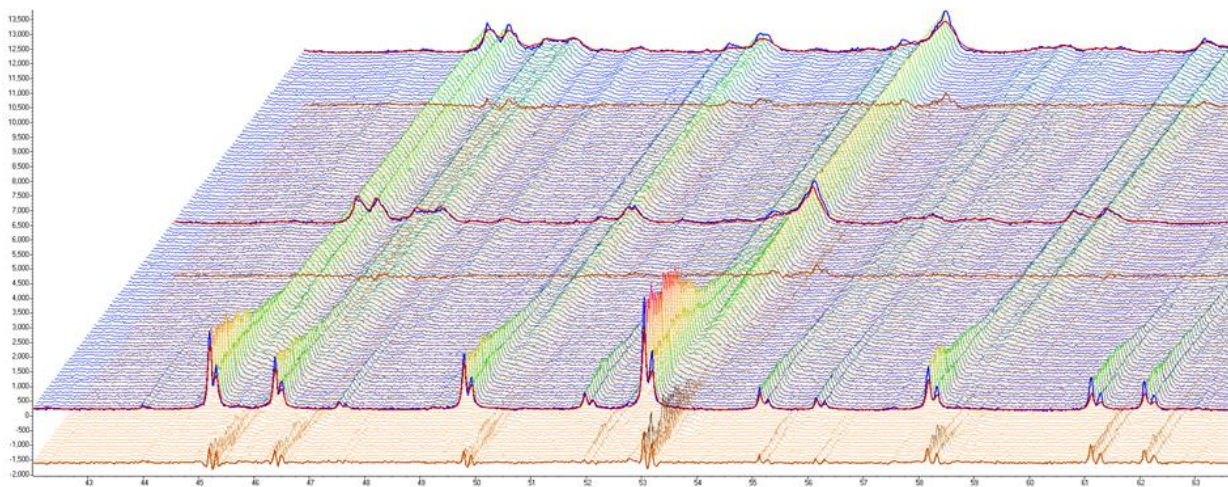
Pressing the Middle Mouse Button and moving the mouse changes the x and y offsets. This movement greatly assists in determining the curvature of the surface. The QuickZoom display is not offset allowing for two views of the same data.

22.1.2 ....Inserting peaks and identifying scans

Peaks can be inserted by pressing the Ctrl-Key and clicking the RMB. When the Ctrl key is pressed a solid circle is displayed on the scan closest to the mouse. The circle is coloured to match the scan lines and in addition the closest scan is displayed with a thickened line. Displayed at the bottom of the plot is the name of the scan as seen by the arrow below. Peaks as well as excluded regions move with the offsets.
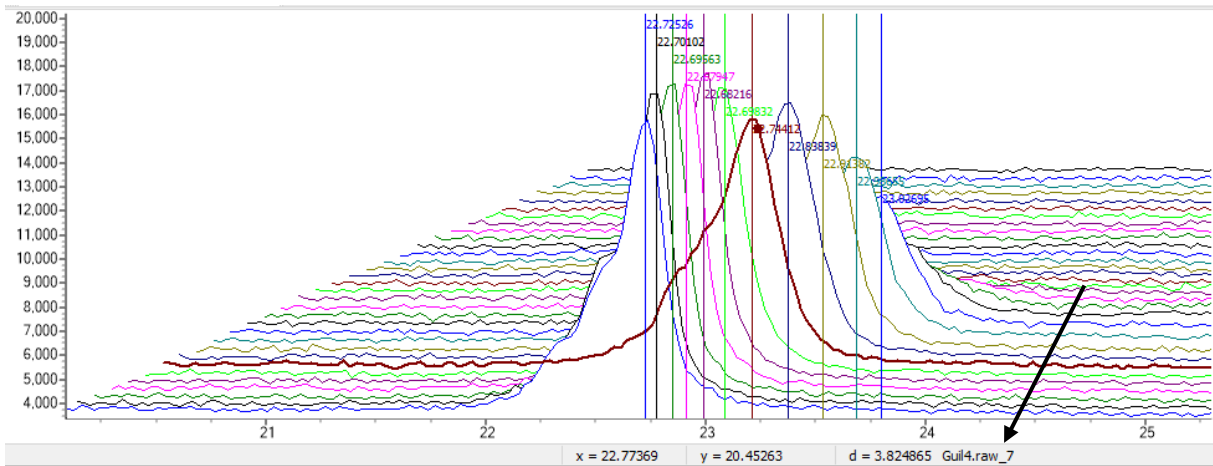
When the Ctrl-Key is pressed the x and y axis values displayed on the status line are offset to match the closest scan. Similarly, when the "For LAM Cursor" option is selected the LAM cursor is changed to match the axis of the closest scan.

22.1.3 ....2D-offset Surface plots

2D-offset plots can be displayed as a 3D-Surface, for example:
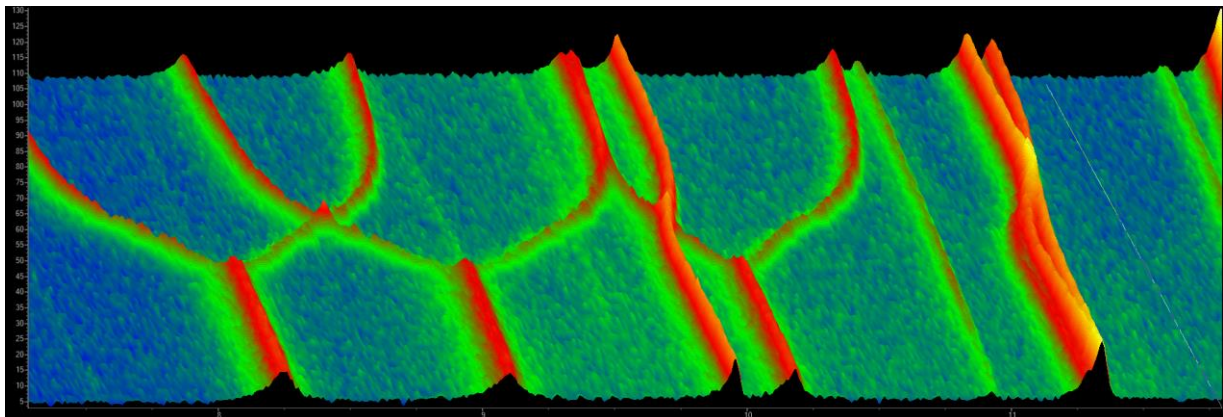
These plots can be manipulated in real time; the 871 file TEST_EXAMPLES\JE-PARA\D8_02999.RAW with over 4 million data points can be easily manipulated:
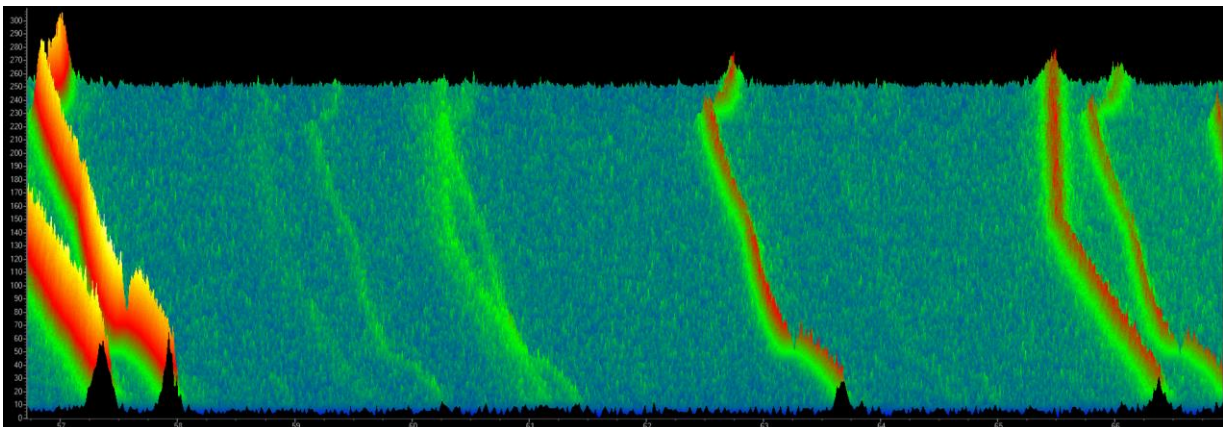
Pressing the Shift key whilst performing a Zoom (forming a box using the mouse) zooms into a region. Zooming in this manner deselects scans for display. An unzoom is performed by

performing an Unzoom whilst holding down the Shift key. Colour schemes can be changed by using the Colours options:
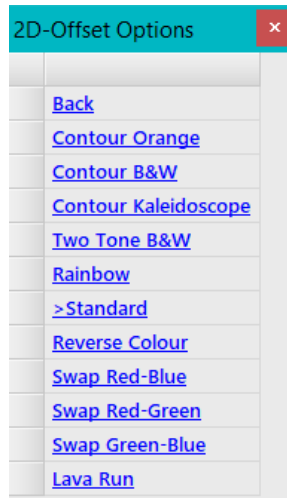
Contour-Orange-15 looks like:
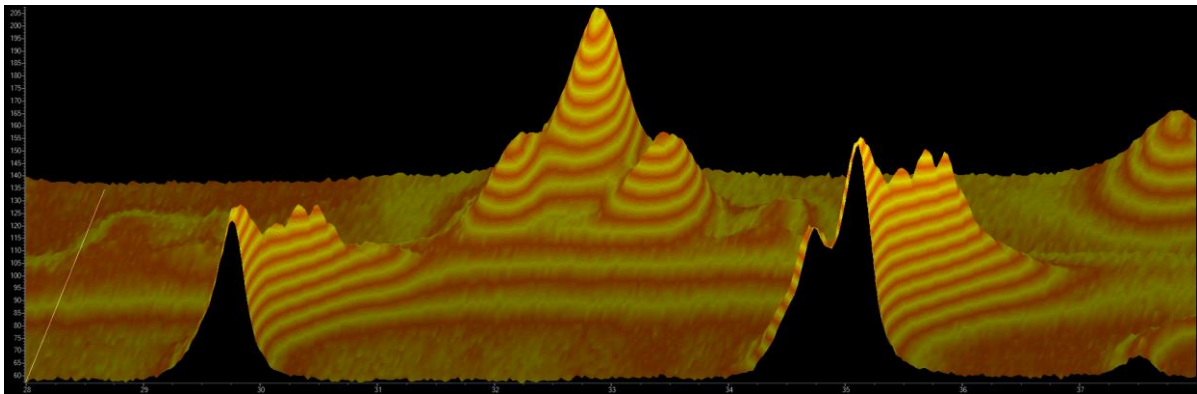
22.1.4 .... 2D-offset Planview plots

Moving the y-offset such that it's at a maximum automatically produces a Planview; a Kaleidoscope colour scheme gives:
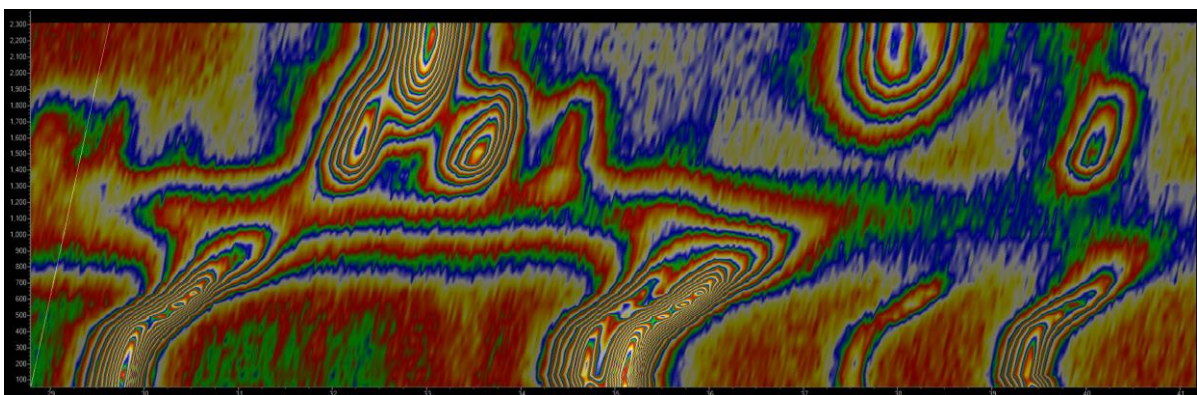
The Standard colour scheme gives:
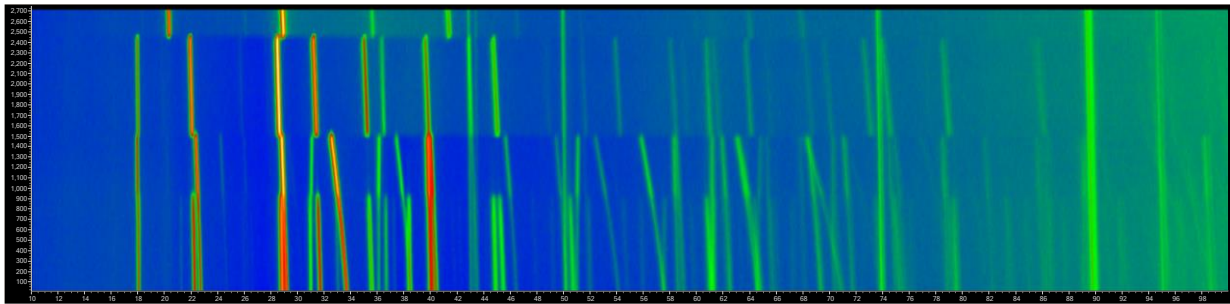
Zooming gives:
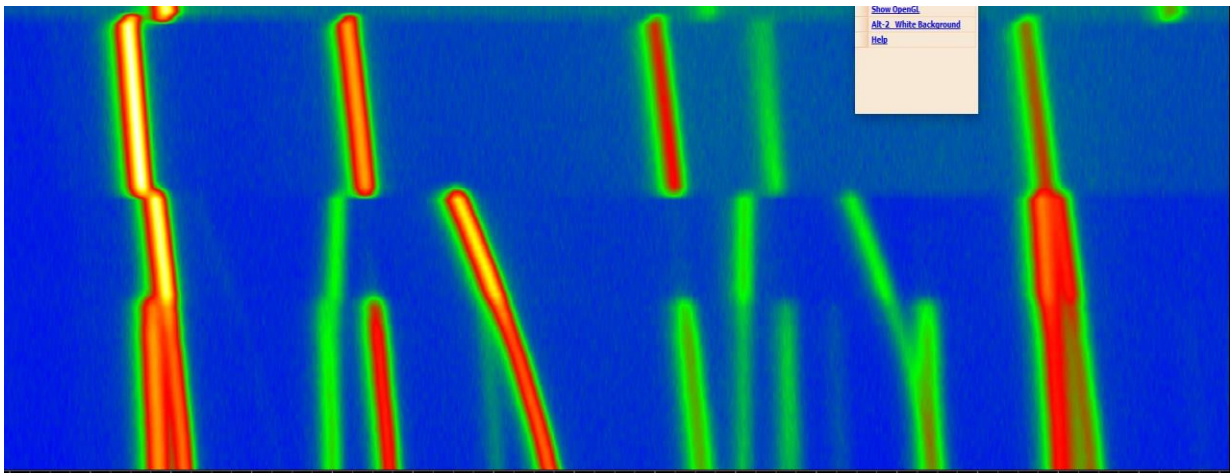
Planview can also have x-axis offsets with line scans overlain:
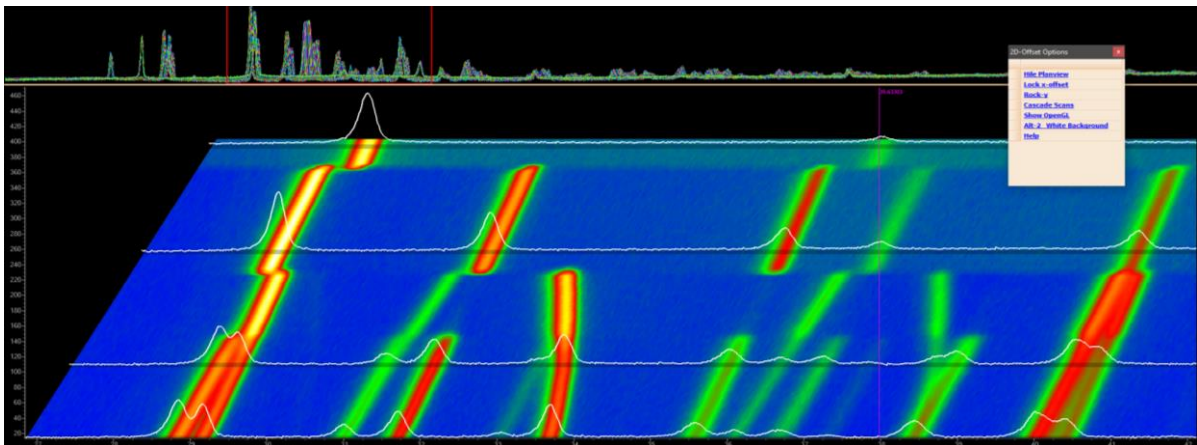
These line scans can include the calculated and/or difference patterns as well as patterns for individual phases. Beneath the displayed line scans are their shadows. Colours are blended across scans as well as across the x-axis to sharpen images.
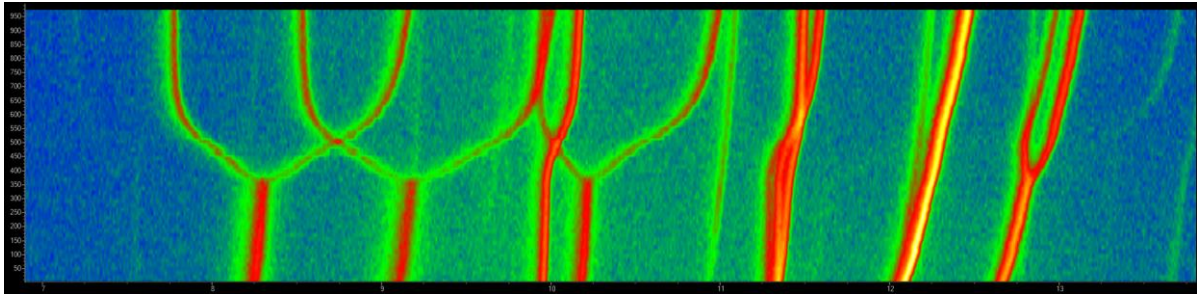
22.1.5 ....OpenGL Surface plots

OpenGL surface plots can be displayed alongside 2D-offset plots:
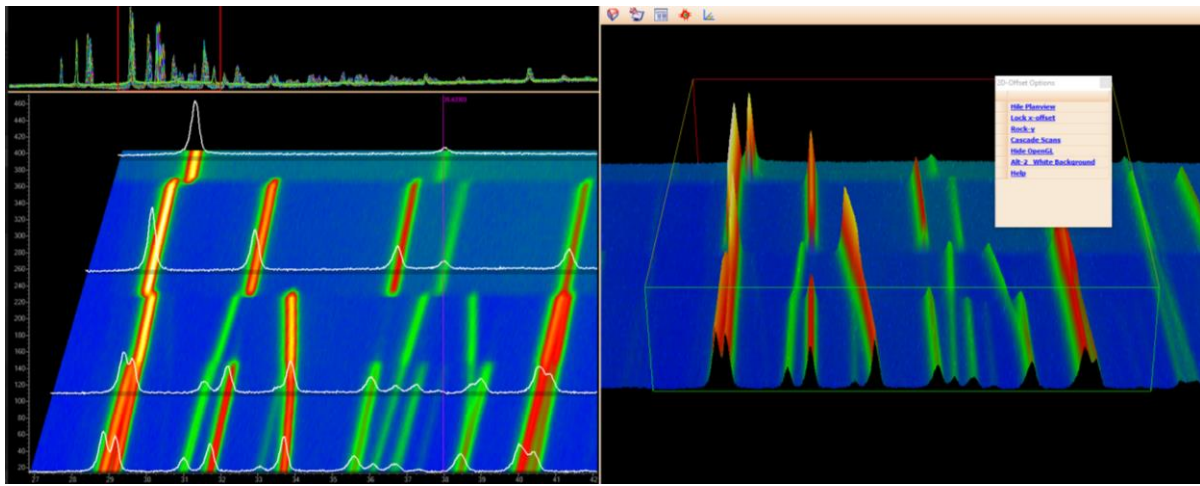
The scans displayed in the chart area are displayed to the right as a surface plot. Use RMB on the surface plot for options; these are:
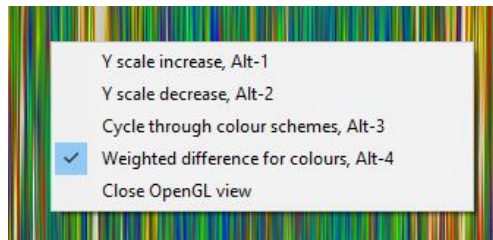
The OpenGL surface plot respects the 2D x-axis and y-axis display options. It is also aware of the QuickZoom window and scrolling. Scrolling can be performed from either the 2D or 3D displays using the Mouse Wheel. Navigation in the OpenGL window is as follows:

• Use the Moise Wheel to scroll the x-axis from either the 2D or 3D plots.

• RMB-Pressed and moving zooms.

• Pressing 'x' whilst rotating allows rotation around an axis vertical to the screen.

• Pressing 'y' whilst rotating allows rotation around an axis horizontal to the screen.

• Pressing 'z' whilst rotating allows rotation around an axis perpendicular to the screen.

• Pressing the Mouse Wheel button (as opposed to rotating the mouse wheel) moves the object and hence the centre of rotation.

• When the Mouse is close to the Left or Right borders of the OpenGL window then rotation is around an axis perpendicular to the computer screen. Very useful for positioning 3D objects.

Opening the OpenGL Text Dialog and clicking on the 3D surface writes text into the Text Dialog; this text comprises the names of the two files bordering the polygon that has been clicked and the average x and y values of the polygon, for example:

22.1.6 ....OpenGL – Weighted difference for colours

The RMB "Weight difference for colours" option displays colours corresponding to:

$$\text{WtDiff} = \text{Abs}(Yobs-Ycalc) \,/\, \text{Weighting}$$

22.2 ... Normalizing scans within a Scan Window

Displayed scans can be normalized using the option "*Yobs* Normalize" which is activated using the RMB on the Scan window. Normalizing scales displayed scans such that the maximum values of the displayed data are all equal. Normalizing is temporary and can be toggled on/off by executing the "*Yobs* Normalize". The following shows scans normalized with all the peaks on the right having the same height.
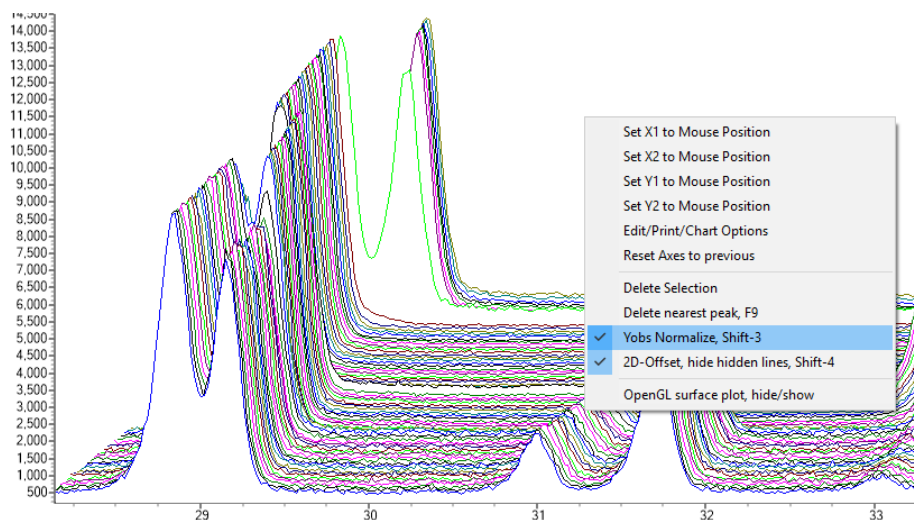
22.3 ... Plotting phases above background

[*fit_obj E* [*min_X* !E] [*max_X* !E]]...
    [*fo_transform_X* !E]
    [*fit_obj_phase* !E]

By default, phases are plotted on top of back ground where back ground comprises *fit_obj*'s+*bkg*. The *xdd* dependent *gui_add_bkg* and the *fit_obj* dependent *fit_obj_phase* can be used to change the defaults, for example,

```
xdd ...
    gui_add_bkg !E
    fit_obj ...
        fit_obj_phase !E
```

gui_add_bkg defaults to 1; if it's zero then phases are not plotted above background. *fit_obj_phase* defaults to 1. If *gui_add_bkg*=1 then the following is added to phases:

> *bkg* + (and any *fit_obj*'s that has *fit_obj_phase* =1)

QUANT\QUANT-7.INP shows the use of *fit_obj_phase*=1 where a *fit_obj* that is a function of a *user_y* object, that is supposed to be a phase, is plotted on top of back ground using a *dummt_str*; the *dummy_str* checks the status of the *fit_obj*'s *fit_obj_phase*.

22.4 ... Plotting fit_objs

fit_obj's can be plotted using the following macros:

```
macro Plot_Fit_Obj(p, name) {        macro Plot_Fit_Obj(name) {
    dummy_str                            dummy_str
        phase_name name                      phase_name name
        scale = p;                       }
}
```

See TEST_EXAMPLES\VOIGT-APPROX\FIT-OBJ.INP for example; i.e.

```
xdd ...
    fit_obj !f1 = ...
    Plot_Fit_Obj(f1, "Fit Obj")
```

Plotting is via a *dummy_str* with the scale parameter set to the name given to the *fit_obj*, which in this case is f1. At the plotting stage the *dummy_str* borrows the calculated pattern from the *fit_obj*. The *scale* parameter of the *dummy_str* has some intelligence built into it such that if *scale* is not a function of a *fit_obj* name then it will search the place of the item it is a function of for a calculated pattern. For example, in the following:

```
xdd ...
    Plot_Fit_Obj(a, "Fit Obj")
    fit_obj = a ...
        prm a ...
```

the 'a' parameter lives locally to the *fit_obj* as it is defined after the *fit_obj*. Defining the scale parameter of the *dummy_str* in terms of 'a' therefore allows the *dummy_str* to determine where to find the calculated pattern to display. In this way macros such as the *PV* macro can be used and plotted without having to define a name for the *fit_obj*, see TEST_EXAMPLES\PVS.INP. Sometimes the *fit_obj* has no name and no parameter that belongs to it; instead of naming the *fit_obj* or rearranging *prm* definitions the second *Plot_Fit_Obj* macro can be used:

```
xdd ...
    fit_obj = ...
    Plot_Fit_Obj("plot previously defined fit_obj")
```
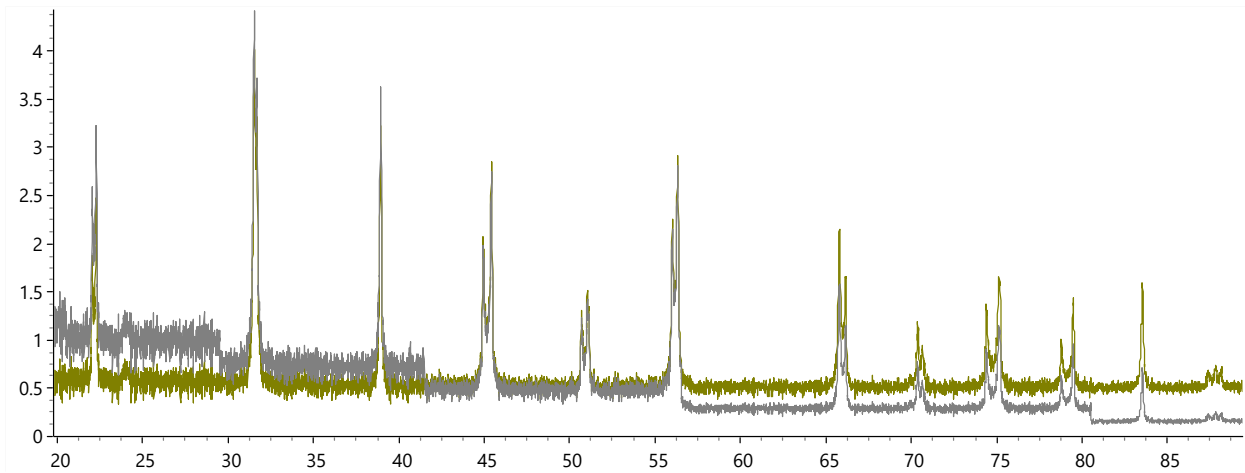
Here the *fit_obj* defined prior to *Plot_Fit_Obj* is plotted.

22.5 ... Display of Normalized SigmaYobs^2

This icon displays normalized *SigmaYobs*^{2}; useful for checking anomalies from VCT or XYE files; here's an example:


The normalization is as follows:

$$\text{SigmaYobs\^{}2 displayed} = \text{SigmaYobs\^{}2 Sum[Yobs] / Sum[SigmaYobs\^{}2]}$$

This puts the display of *SigmaYobs^2* on a similar scale to *Yobs*. For normal x-ray data *SigmaYobs* = Sqrt(*Yobs*) and hence nothing is done as the displayed plot would simply be equal to *Yobs*. On some data sets, TOF for example, the magnitude of *SigmaYobs* can be small; therefore, when refining on multiple data sets from different sources, the weighting schemes may need to be modified in order to give the desired weight to the data sets.

22.6 ... Cumulative Chi2

A kernel operation that results in the following graphical display:
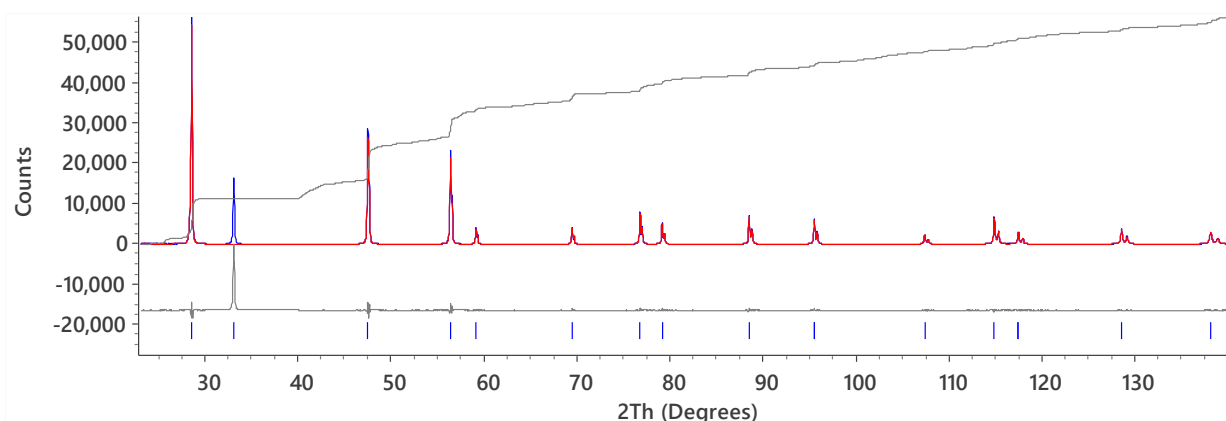
Uses the weighting from the kernel which can be user defined or otherwise. *SigmaYobs* is used in the weighting if it exists. The Cumulative Chi2 it is normalized to have the maximum intensity of *Yobs* within the display window. Data is obtained from the kernel and excluded regions are ignored as shown in the plot above. Tabs for Cumulative Chi2 has been included in the appropriate GUI tabs, ie.
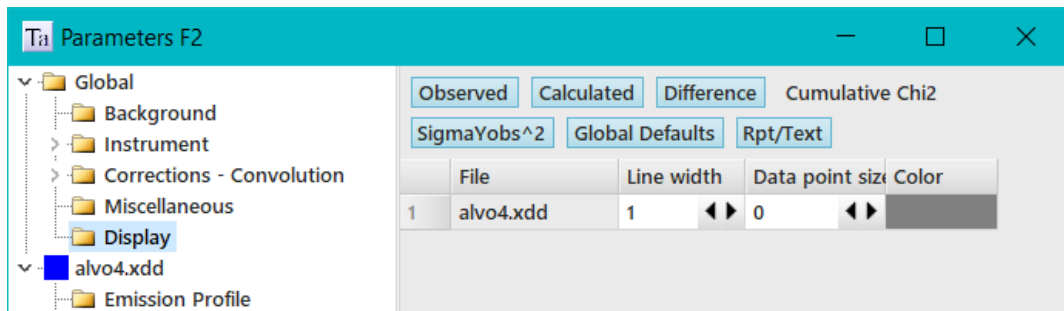
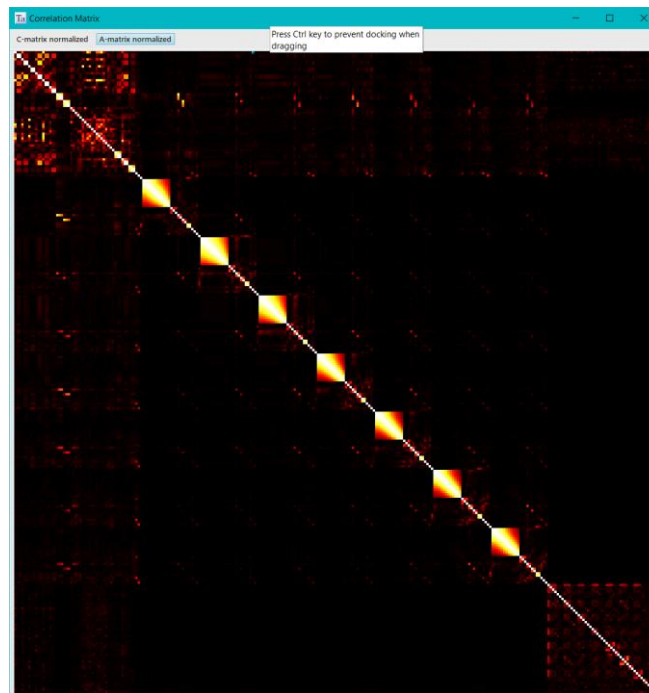## 22.7 ... Correlation Matrix display

A Correlation matrix window activated from the Fit Dialog; it operates in Launch or GUI modes. Example output is as follows:
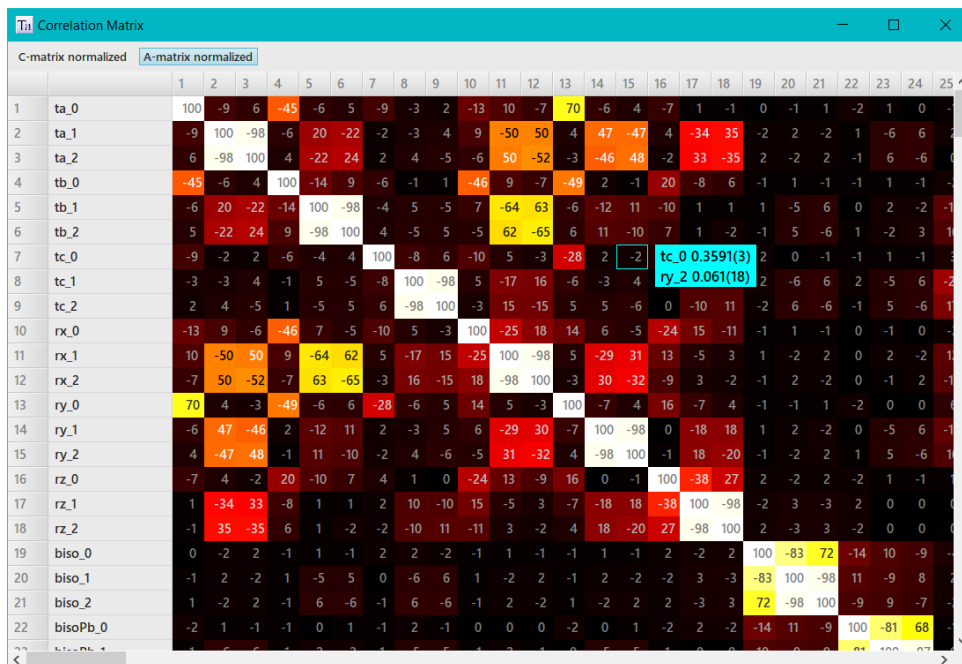
Both the A-matrix and the correlation matrix include penalties/restraints depending on whether *do_errors_include_penalties* and/or *do_errors_include_restraints* are defined. The display of the matrix can be zoomed using Ctrl-MouseWheel, here's an example:
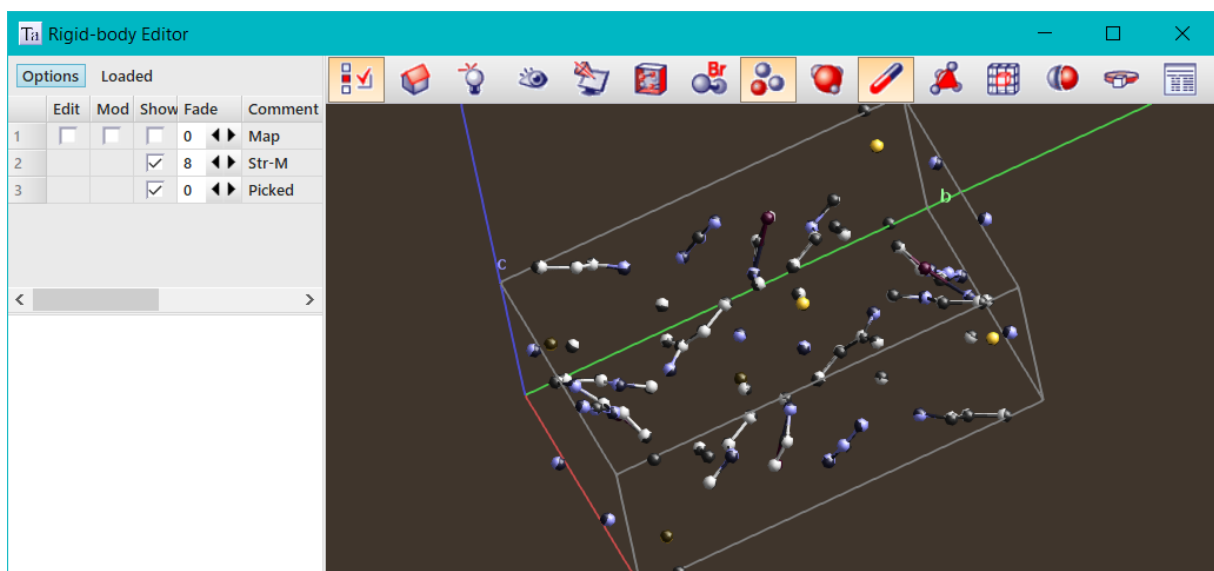
MouseMove over the correlation matrix displays a Hint comprising the corresponding parameter names, values and errors. Left Mouse button down and dragging translates the matrix.

22.8 ... Fading a structure

The intensity of atom colours displayed in OpenGL can be adjusted using the Fade spin button of the OpenGL grid options; for example:
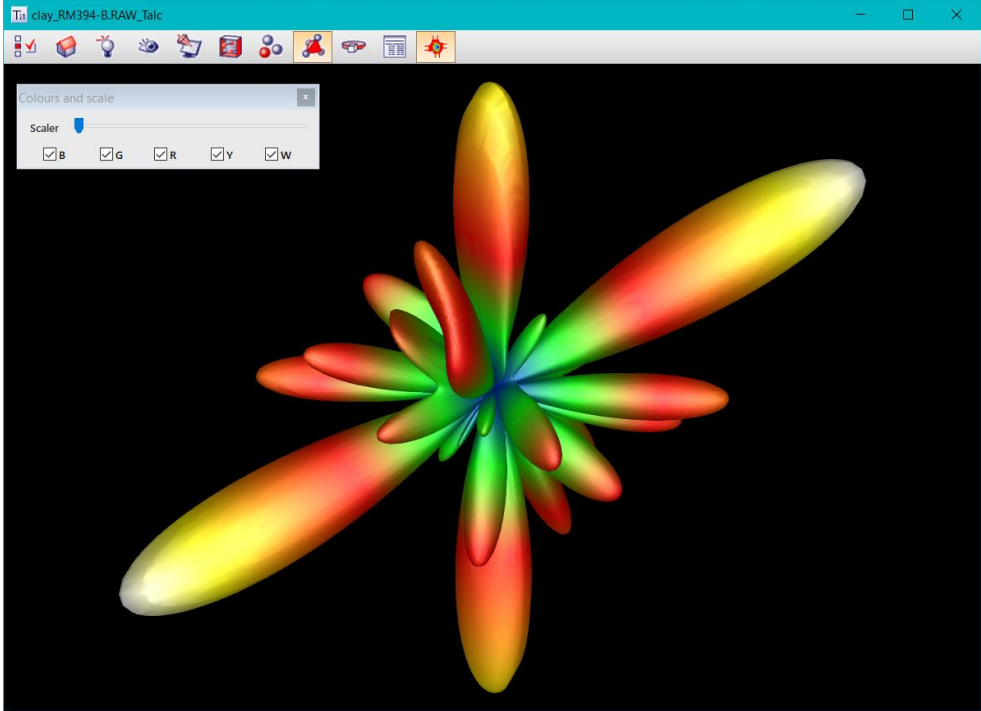

22.9 ... Normals Plot

[*normals_plot* !E]...
    [*normals_plot_min_d* !E]

An OpenGL plot of lattice plane Normals with the lengths of the Normals defined by *normals_plot*. For example:

```
normals_plot = Abs(H * K + L^2) + 1; normals_plot_min_d 0.3
```

normals_plot_min_d is optional; small values (i.e. 0.1) could lead to millions of points and users could blow up their computers. Here's some output from the test example CLAY.INP:

The slider in the plot is activated by clicking on the button. This slider multiplies the length of the *normals_plot* equation before generating the surface. The exact formulation for the multi-plications is as follows.

Definitions:

    s = multiplier which has a value (not shown to the user) that varies from 0 and 1.

    N = diffraction vector directions with lengths given by the *normals_plot* equation.

    N = Sqrt(**N** . **N**) = magnitude of **N**

    N_{max} = maximum N

Before generating the shape, **N** is multiplied by:

For $s < 0.25$	: $((N / N_{max})^{(4*s)}) * N_{max} / N$
For $0.25 \leq s < 0.5$	: $((N / N_{max})^{(4*(0.5-s))}) * N_{max} / N$
For $0.5 < s \leq 0.75$	: $(4*(0.5-s)) * N_{max} / N + (1-s)$
For $0.75 < s \leq 1$	: $(((4*(0.75-s)) + N_{max} / N$

22.10 . Improvements to the Grid

Data can be sorted by double clicking on column headings. Sorting alternates between ascending and descending order. On leaving a grid the column most recently sorted is remembered. On re-entry of that grid the data is again sorted according to the saved state. A small < or > sign is

displayed to the left of the column heading name. Sorting works for all grids that display data with rows that are similar in Type; i.e. Peak data, sites etc... Val and Error columns are sorted numerically. Hkls, F^2 and other obvious numeric columns are also sorted numerically. However, Min and Max are sorted using strings as they can be equations and hence their fields are strings.

CTRL-MouseWheel zooms/un-zooms the text of a grid.

MouseDownMouseMove for Panning.

22.11 . Mouse operation in OpenGL Graphics

First some definitions

LMB = Left Mouse Button

RMB = Right Mouse Button

MID = Mouse Wheel or Middle button on Laptops

MM = Mouse Moving

WM = Wheel moving

LMB-D = Left Mouse Button Down

RMB-D = Right Mouse Button Down

- MW-D = Mouse Wheel Down
- For example, LMB-D-MM is simply dragging with the LMD

Image rotation/translation operations are:

- LMB-D- MM rotates the image.
- LMB-D- MM and quick release initiates continuous rotation.
- LMD-D-MM on the first 10% of the viewport from the left or the last 10% from the right rotates around an axis perpendicular to the screen. This is another way of doing what Shift-LMB-D-MM does but without the need for keyboard input. 10% seems a good amount as it does not seem to interfere with normal rotation.
- MW zooms in addition to the usual RMB-D-MM.
- MID-D-MM translates the image in the plane of the screen.

Images are rotated around the centre of gravity (or centre of unit cell) unless there's a change using the RMB-D options.

23. REFERENCES

Ainsworth, C. M.; Lewis, J. W.; Wang, C.; Johnstone, H. E.; Mendhis, B. G.; Brand, H. E. A.; Coelho, A. A.; Evans, J.S.O. (2016). *Chem. Mater.* **28**, 3184–3195. *"3D Transition Metal Ordering and Rietveld Stacking Fault Quantification in the New Oxychalcogenides $La_2O_2Cu_{2-4x}Cd_{2x}Se_2$"*

[a]Baerlocher, C; Gramm, F.; Massüger, L; McCusker, L; He, Z; Hovmöller, S & Zou, X. (2007). *SCIENCE* VOL **315** 23 FEBRUARY 2007

[b]Baerlocher, C.; McCusker, L. B.; & Palatinus, L. (2007). Z. Kristallogr. **222**, 47–53

Balzar, D. (1999). Microstructure Analysis from Diffraction, edited by R. L. Snyder, H. J. Bunge, and J. Fiala, International Union of Crystallography, 1999. *"Voigt-function model in diffraction line-broadening analysis"*

Bergmann, J., Kleeberg, R., Haase, A. & Breidenstein, B. (2000). *Mat. Sci. Forum*, 347-349, 303-308. *"Advanced Fundamental Parameters Model for Improved Profile Analysis".*

Brindley, G. W. (1945). *Phil. Mag.* **36**, 347-369. *"The effect of grain or particle size on X-ray reflections from mixed powders and alloys, considered in relation to the quantitative determination of crystalline substances by X-ray methods"*

Broyden, C. G. (1970). *J. Inst. Maths. Appl.*, **6**, 76-90. *"The Convergence of a Class of Double-rank Minimization Algorithms"*

Cagliotti, G., Paoletti, A. & Ricci, F. P. (1958). *Nucl. Inst.* **3**, 223-228. *"Choice of collimators for a crystal spectrometer for neutron diffraction"*

Cheary, R. W. & Coelho, A. (1992). *J. Appl. Cryst.* **25**, 109-121. *"A fundamental parameters approach to X-ray line-profile fitting"*

Cheary, R. W. & Coelho, A. A. (1994). *J. Appl. Cryst.* **27** (5), 673-681. *"Synthesizing and fitting linear position-sensitive detector step-scanned line profiles"*

Cheary, R. W. & Coelho, A. A. (1998a). *J. Appl. Cryst.* **31**, 851-861. *"Axial divergence in a conventional X-ray powder diffractometer I. Theoretical foundations"*

Cheary, R. W. & Coelho, A. A. (1998b). *J. Appl. Cryst.* **31**, 862-868. *"Axial divergence in a conventional X-ray powder diffractometer II. Implementation and comparison with experiment"*

Cheary, R. W.; Coelho, A. A. and Cline, J. P. (2004). *Journal of Research-National Institute of Standards and Technology*, **109** (2004): 1-26. *"Fundamental parameters line profile fitting in laboratory diffractometers"*

Coelho, A. A. & Cheary, R. W. (1997). *Computer Physics Communications*, **104**, 15-22. *"A fast and simple method for calculating electrostatic potentials"*

Coelho, A. A. (2000). J. Appl. Cryst. **33**, 899-908, *"Structure Solution by Simulated Annealing"*

Coelho, A. A. (2003). *J. Appl. Cryst.* **36**, 86–95. *"Indexing of powder diffraction patterns by iterative use of singular value decomposition".*

Coelho, A. A. (2005). *J. Appl. Cryst.* **38**, 455–461. "*A bound constrained conjugate gradient solution method as applied to crystallographic refinement problems*"

Coelho, A. A. (2007). *Acta Cryst.* **A36**, 400–406. *"A charge-flipping algorithm incorporating the tangent formula for solving difficult structures"*

Coelho, A. A; Evans, J.; Evans, I; Kern, A.; Parsons, S. (2011). *Powder Diffraction*, Vol. **26**, Issue S1, pages S22–S25, *"The TOPAS symbolic computation system"*

Coelho, A. A.; Chater, P.A.; Kern, A. (2015). *J. Appl. Cryst.* **48**, Part 3, 869–875. *"Fast synthesis and refinement of the atomic pair distribution function"*

Coelho, A. A.; Evans, J. S. O. & Lewis, J. W. (2016). *J. Appl. Cryst.* **49**, 1740-1749. "*Averaging the intensity of many-layered structures for accurate stacking-fault analysis using Rietveld refinement*"

[a]Coelho, A. A. & Rowles, M. R. (2017). *J. Appl. Cryst.* **50**, 1331-1340. "*A capillary specimen aberration for describing X-ray powder diffraction line profiles for convergent, divergent and parallel beam geometries*". https://doi.org/10.1107/S160057671701130X.

[b]Coelho, A. A. (2017). *J. Appl. Cryst.* **50**, 1323-1330. "*An indexing algorithm independent of peak position extraction for X-ray powder diffraction patterns*". https://doi.org/10.1107/S1600576717011359.

[a]Coelho, A. A. (2018). *J. Appl. Cryst.* **51**, 112-123. "*Deconvolution of instrument and $K\alpha$ contributions from X-ray powder diffraction patterns using least squares with penalties*". https://doi.org/10.1107/S1600576717017988.

[b]Coelho, A. A. (2018). *J. Appl. Cryst.* **51**, 210-218. "*TOPAS & TOPAS-Academic: An optimization program integrating computer algebra and crystallographic objects written in c++*". https://doi.org/10.1107/S1600576718000183

[c]Coelho, A. A. (2018). *J. Appl. Cryst.* **51**, 428-435. *"Optimum Levenberg-Marquardt constant determination for nonlinear least-squares"*

Baerlocher, C; Gramm, F.; Massüger, L; McCusker, L; He, Z; Hovmöller, S & Zou, X. (2007). *SCIENCE* VOL 315 23 FEBRUARY 2007.

Burla, C.B; Carrozzini, B.; Cascarano, G. L.; Giacovazzo C. & Polidori, G. (2011). J. Appl. Cryst. 44, 1143–1151

Chernick, M. R. (1999). *Bootstrap Methods, A Practitioner's Guide*, Wiley, New York.

David, W. I. F; Matteo, L.; Scardi, P. (2010). *Materials Science Forum Vol. 651 pp 187-200*

DiCiccio, T. J. & Efron, B. (1996). Bootstrap confidence intervals (with discussion), *Statistical Science* **11**, 189–228.

Durbin, J. & Watson, G. S. (1971). *Biometrika.* **58**, 1-19. *"Testing for Serial Correlation in Least Square Regression, III"*

Efron, B. & Tibshirani, R. (1986). Bootstrap methods for standard errors, confidence intervals and other measures of statistical accuracy, *Statistical Science* 1, 54–77.

Favre-Nicolin, V; Cerny, R. (2002). *J. Appl. Cryst.* **35** (6), 734-743.

Fletcher, R. (1970). *Comput. J.*, **13**, 317-322. "*A New Approach to Variable Metric Algorithms*"

Finger, L. W., Cox, D. E & Jephcoat, A.P. (1994). *J. Appl. Cryst.* **27**, 892-900. "*A correction for powder diffraction peak asymmetry due to axial divergence*"

Flack, H. D. (1983). *Acta Cryst.* **A39**, 876-881

Goldfarb, D. (1970). *Math. Comp.*, **24**, 23-26. "*A Family of Variable Metric Updates Derived by Variational Means*"

Hauptman, H. & Karle, J. (1956). *Acta Cryst.* **9**, 635

Hill, R. J. & Flack, H. D. (1987). *J. Appl. Cryst.* **20**, 356-361. "*The Use of the Durbin-Watson d Statistic in Rietveld Analysis*"

Hölzer, G., Fritsch, M., Deutsch, M., Härtwig, J. & Förster, E. (1997). *Physical Review A*, **56,** 4554-4568. "*K$\alpha_{1,2}$ and K$\beta_{1,2}$ X-ray emission lines of the 3d transition metals*"

Järvinen, M. (1993). *J. Appl. Cryst.* **26**, 525-531. "*Application of symmetrized harmonics expansion to correction of the preferred orientation effect*"

Johnson, C. K. & Levy, H. A. (1974). International Tables for X-ray Crystallography, **IV**, 311 - 336. "*Thermal-motion analysis using Bragg diffraction data*"

Kopp, Joachim. (2006). *Int.J.Mod.Phys.* **C19**:523-548,2008

Leoni, M.; Di Maggio, R.; Polizzi, S; Scardi P. (2004), *J. Am. Ceram. Soc.* **87**, 1133-1140.

Lister, S. E.; Radosavljevic Evans, I.; Howard, J. A. K.; Coelho A. and Evans, J. S. O. (2004). *Chemical Communications*, Issue **22**.

March, A. (1932). *Z. Krist.* **81**, 285-297. "*Mathematische Theorie der Regelung nach der Korngestalt bei affiner Deformation*"

Markvardsen, A. J.; David, W. I. F.; Johnston, J. and Shankland K. (2001), *Acta Cryst.* **A57**, 47

Marquardt, D. W. (1963). *J. Soc. Ind. Appl. Math.* **11**(2), 431-331. "*An algorythm for least-squares estimation of nonlinear parameters*"

Mooers, B. H. M. (2016). Acta Cryst. D72, 477–487

O'Connor, B. H.; and Raven, M. D. (1988). *Powder Diffraction*, Vol. **3**, No. 1. "*Application of the Rietveld Refinement Procedure in Assaying Powdered Mixtures*"

Oszlányi, G. & Süto A. (2004). *Acta Cryst.* A**60**, 134-141

Oszlányi, G. & Süto A. (2005). *Acta Cryst.* A**61**, 147-152

Oszlányi, G. & Süto A. (2006). *Acta Cryst.* A**63**, 156–163

Oszlányi, G.; Süto A.; Czugler, M. & Parkanyi, L. (2006). *J. AM. CHEM. SOC.* 9 VOL. 128, NO. 26, 8393. "*Charge Flipping at Work: A Case of Pseudosymmetry*".

Pawley, J. S. (1981). *J. Appl. Cryst.* **14**, 357

Rietveld, H. M. (1969). *J. Appl. Cryst.* **2**, 65–71.

Scardi, P. & Leoni, M. (2001). *Acta Cryst. A* **57**, 604–613.

Shanno, D. F. (1970). *Mathematics of Computing*, Vol. **24**, pp 647–656. "*Conditioning of Quasi-Newton Methods for Function Minimization*"

Favre-Nicolin, V. and Cerny, R. (2002) EPDIC 8 proceedings. "*Fox: Modular Approach to Crystal Structure Determination from Powder Diffraction*"

Sabine, T. M., Hunter, B. A., Sabine, W. R., Ball, C. J. (1998): *J. Appl. Cryst.* **31**, 47–51

Schneider, T. R. & Sheldrick, G. M. (2002). *Acta Cryst.* D**58**, 1772–1779. "*Substructure solution with SHELXD*"

Shiono, M. & Woolfson, M. M. (1992). *Acta Cryst.* A**48**, 451–456

Whitfield, P. S. and Coelho, A. A. (2016). *J. Appl. Cryst.* **49**, 1806–1809. "*Asymmetric band flipping for time-of-flight neutron diffraction data*".

Young, R. A. (1993). The Rietveld Method, edited by R.A. Young, IUCr Book Series, Oxford University Press 1993, 1–39. "*Introduction to the Rietveld method*"