What's New

TOPAS-Academic

Version 8

Alan A Coelho www.topas-academic.net

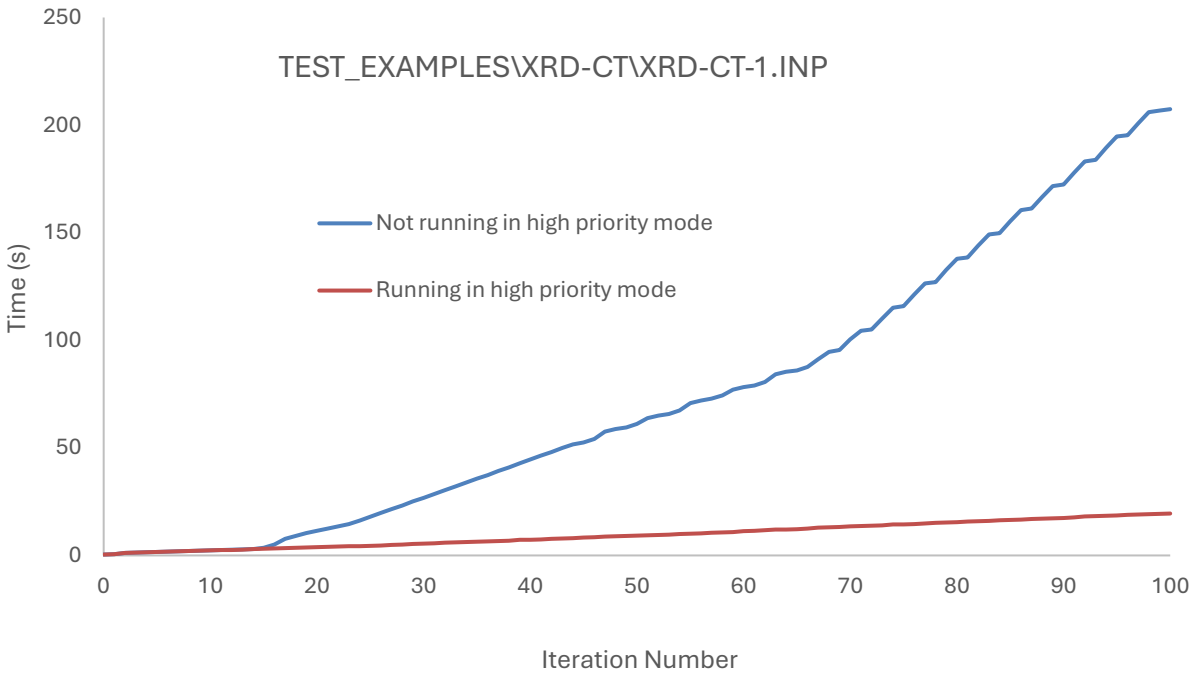
November 7, 2025

Contents

1. ..NEW FUNCTIONALITY

1.1..... Running TOPAS in high priority mode

Windows is becoming more guarded with the compiler not producing EXE files that run in a high priority mode. This slows down TOPAS appreciably when running large refinements, especially when accompanied by large memory usage. The solution is to run the program in high-priority-mode. Below shows a factor of 10+ difference in running-time when running XRD-CT-1.INP using TC.EXE on Window 11.



When not running in high priority mode, time per iteration slows downs appreciably after iteration 16.

1.1.1 ..... Running TA.EXE in high priority mode (TOPASH.BAT)

Run the following from the command line:

```
start "" /high "ta"
```

Or, run the batch file TOPASH.BAT.

1.1.2 ..... Running TC.EXE in high priority mode

From the command prompt, use the following to start the command prompt:

```
start /high "cmd"
```

Or, run the command as an administrator from the start menu by typing ‘command’ and then choose the “Run as administrator”.

1.2..... New PDF functionality

1.2.1 ..... ADPs in PDF refinement

```
site... occ Zr 1 u11 @ .01  u22 @ .01  u33 @ .01  u12 @ 0  u13 @ 0  u23 @ 0
    adps_scale @ 1
```

ADPs can now be used and refined in PDF refinement. The syntax is similar to reciprocal space refinement where the apds keyword, when used, generates the ADP parameters, for example, the following:

```
site Zr1 x # y # z # occ Zr 1 apds
```

becomes:

```
site Zr1 x # y # z # occ Zr 1 ADPs { u11 # u22 # u33 # u12 # u13 # u23 # }
```

This implementation is similar to PDFGui (Farrow *et al.*, 2007) where peaks are Gaussian even at low-*r*. ADP parameters will therefore correct for peak width but not asymmetry. Asymmetry does occur however and is noticeable when atomic displacement geometry is extreme.

adps_scale allows for the scaling of the Uij parameters and it can be a function of *X* where *X* corresponds to the distance between atoms.

```
prm !delt1 0.75 min 1e-6 max 5
prm !delt2 0 min 1e-5 max 1
prm !Qb  0.05 min 1e-6 max 1
prm aa 1 min 1 _v = Rand(0.5, 1.5);
adps_scale = 2 aa (Abs(1 - delt1 / X - delt2 / X^2 + (Qb^2) X^2));
```

The FWHM of a PDF peak for atom *i* and *j* is given by:

$$FWHM_{ij} = Sqrt(adp\_scale_i\ U_{cart,i} + adp\_scale_j\ U_{cart,j})$$

where U_{cart} is U_{ij} in cartesian coordinates.

1.2.2 ..... Multiatom approach to ADPs in PDF refinement

macro ADP_5 and ADP_7 See file PDF-ADPS.INC	**Examples** TEST_EXAMPLES\PDF-ADPS\ APPROX-1.INP FIT_TO_GR.INP

In many cases, anisotropic displacement parameters in PDF refinement can be described using 5 or 7 beq type sites, we will call these descriptions ADP_5 and ADP_7. ADP_5 comprises 7 parameters instead of the normal 6 unn parameters. These ADP_5 parameters can be transformed to unn parameters by fitting unn parameters to a pattern created from the ADP_5

parameters. The fit is reasonable considering the unn model has only 6 parameters. Some main points when using ADP_5 in PDF refinement:

- Number of ADP parameters become 7 instead of 6.
- Broadening due to ADPs in G(r) is implied.
- Asymmetry at low r is implied.
- This approach works in Version 7 (albeit slower)

Asymmetry seen at low r is typically difficult to model; the ADP_n approach however implicitly contains asymmetry. The computational effort increases as there are 7 atoms per ADP site; this is offset by the very fast calculation of PDF patterns using beq type sites. The file PDF-ADPS.INC contains the macros necessary for describing ADPs-7. APPROX-1.INP demonstrates the ability of the ADPs-7 approach to describe unn models in reciprocal space. It has three modes of operation:

1) CREATE_USING_unns: creates a simulated single crystal pattern from normal unn parameters for one atom. neutron_data is used the effects of atomic scattering factors.
2) FIT_USING_ADPs_5: fits to the simulated pattern using the ADPs-7 approach. This refinement then saves the calculated ADPs-7 pattern created to a file called SIM-2.HKL.
3) DETERMINE_unns_FROM_ADPs_5: fits normal unn parameters to SIM-2.HKL.

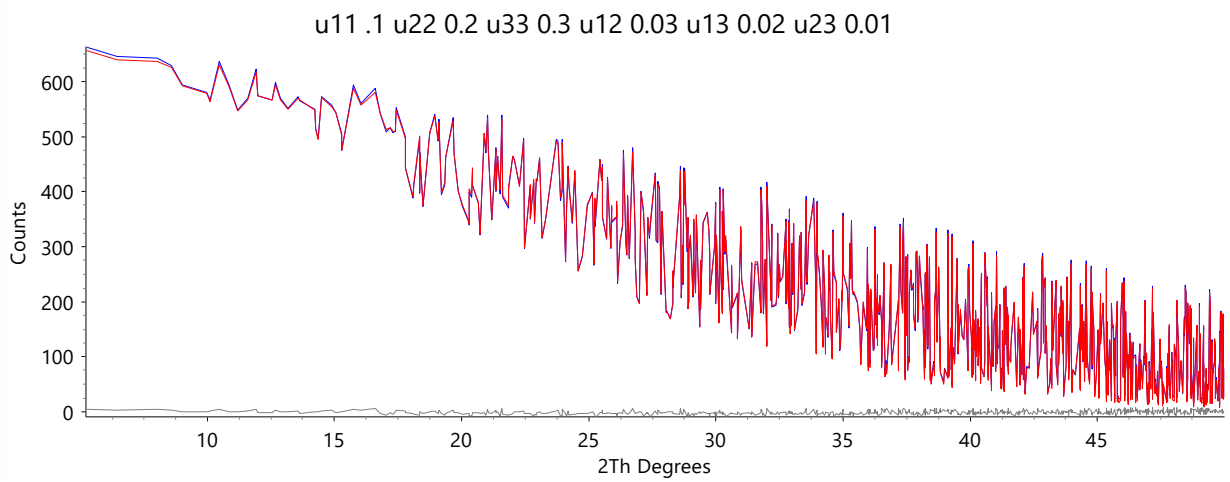
ADP_5 sites are described in the ADP_5 macro, and it looks like:

```
macro ADP_5_0(s, &x0, &y0, &z0, atom, &o, &x1, &y1, &z1, &x2, &y2, &z2, &bo)
{
site s x = x0;  y = y0;  z = z0;  occ atom = 0.2 o; beq = bo;
local #m_unique ns = Get(num_posns);
site s##_1p x = x0+x1; y = y0+y1; z = z0+z1; occ atom = 0.2 (ns / Get(num_posns)) o; beq = bo;
site s##_2p x = x0+x2; y = y0+y2; z = z0+z2; occ atom = 0.2 (ns / Get(num_posns)) o; beq = bo;
site s##_1m x = x0-x1; y = y0-y1; z = z0-z1; occ atom = 0.2 (ns / Get(num_posns)) o; beq = bo;
site s##_2m x = x0-x2; y = y0-y2; z = z0-z2; occ atom = 0.2 (ns / Get(num_posns)) o; beq = bo;
}
```

Two extreme cases have been performed; results for the first case, the refined and original Uij parameters are:

```
   ADPs {  0.39524 0.39690  0.40143 -0.18277 -0.19009 -0.19268 } ' refined
   ADPs {  0.4     0.4      0.4     -0.19    -0.19     -0.19 }   ' original
```

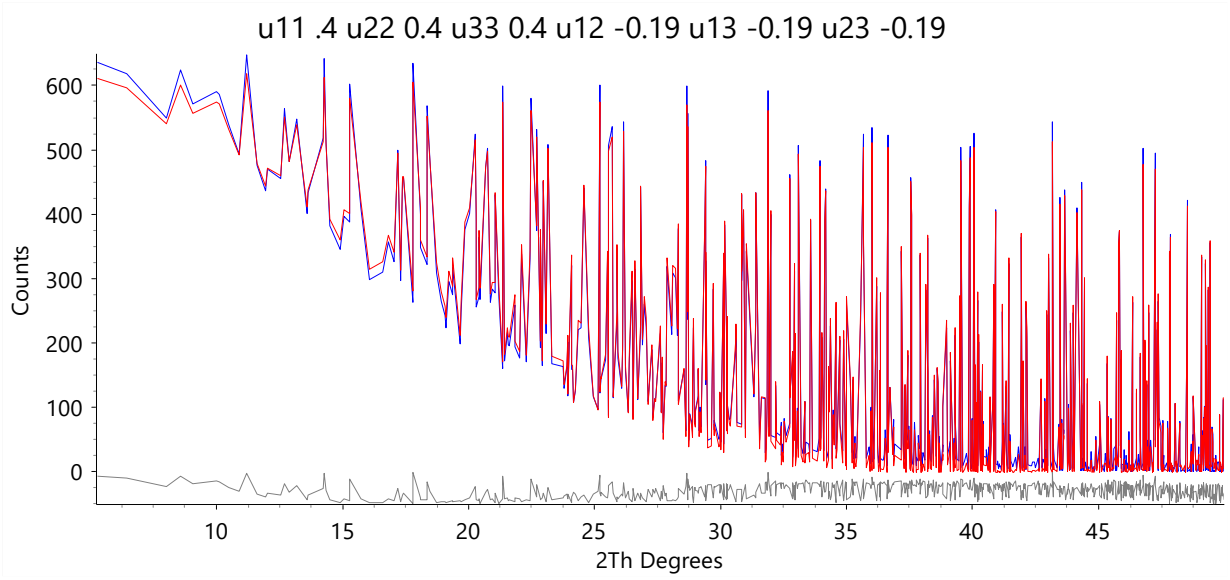
The refined values, from the DETERMINE_unns_FROM_ADPs_7 operation, shows good agreement with the original values. The FIT_USING_ADPs_7 operation produces a fit that looks like:



A further extreme example is:

```
ADPs {  0.39524 0.39690  0.40143 -0.18277 -0.19009 -0.19268 } ' refined
ADPs {  0.4     0.4      0.4     -0.19    -0.19     -0.19 }   ' original
```

The FIT_USING_ADPs_5 operation produces a fit that looks like:



1.2.2.1 ..... Multiatom approach to ADPs – fitting to G(r) patterns

This section generates describes the a reciprocal space pattern using unn parameters, then generates a G(r) pattern from the simulated data. Then fits to the G(r) pattern using either ADP_5 or Uij parameters. Additionally, a reciprocal space patterns can then be simulated using the fitted ADP_5 parameters and then finally the loop is complete with a unns fit to the reciprocal space pattern. The final unn parameters should match the original unn parameters reasonably well. The control parameters are as follows:

```
#prm generate_recip_space_pattern = 1;
#prm generate_Gr_created_from_sine_transform = 0;
#prm generate_Gr_calc_using_Uij = 0;
#prm ADP_5_fit_to_Gr_calc_using_Uij = 0;
#prm ADPs_fit_to_Gr_calc_using_Uij = 0;
#prm ADP_5_fit_to_Gr_created_from_sine_transform = 0;
#prm ADPs_fit_to_Gr_created_from_sine_transform = 0;
#prm create_recip_ADP_5_fit_to_Gr_calc_using_Uij = 0;
#prm create_recip_ADP_5_fit_to_Gr_created_from_sine_transform = 0;
#prm Fit_create_recip_ADP_5_fit_to_Gr_calc_using_Uij = 0;
#prm Fit_create_recip_ADP_5_fit_to_Gr_created_from_sine_transform = 0;
macro Append_to_File_Name { 1 } ' anything here to identify output files created
#prm include_resolution_broadening = 1;
```

These need to be executed one at a time and in sequence; they should be self-explanatory. The main point is that the sine transform pattern created using generate_Gr_created_from_sine_transform comprises asymmetry whereas the calculated G(r) created using generate_Gr_calc_using_Uij does not. The former can be considered the 'true' G(r) pattern. Also of importance is that the pattern created using generate_Gr_created_from_sine_transform is generally better fitted using ADP_5 (or ADP_7) using than when using ADPs_fit_to_Gr_created_from_sine_transform. The reason is that the latter does not include asymmetry.

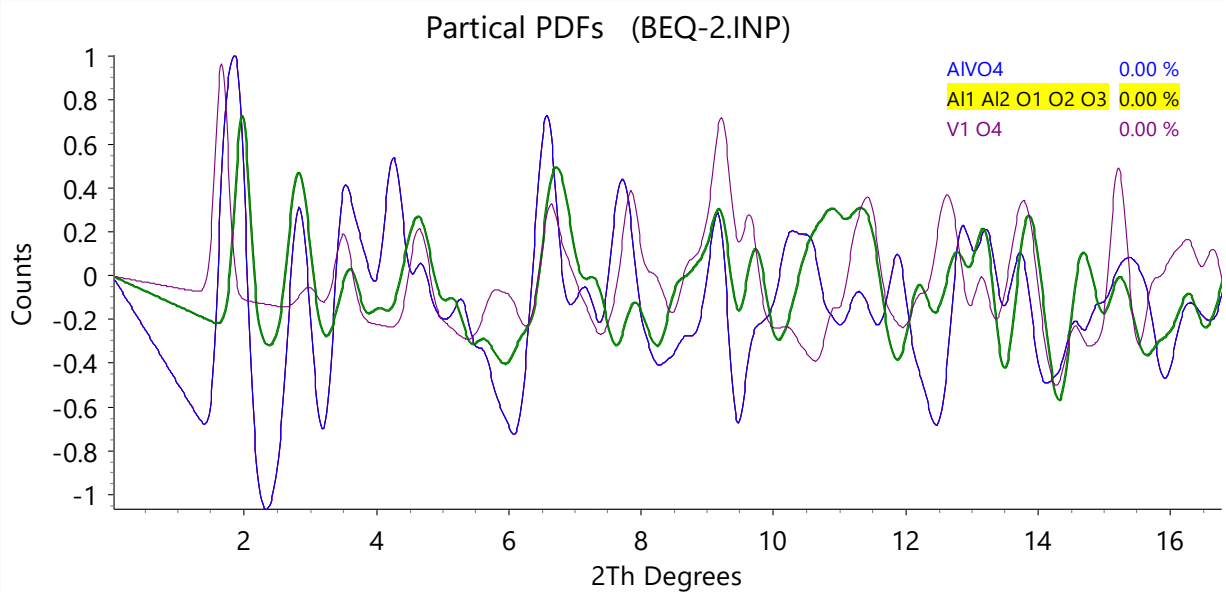
1.3..... Displaying partial PDFs

[str]... [pdf_partial_1 $sites] [pdf_partial_2 $sites] [pdf_partial_when !E1]	**Examples** TEST_EXAMPLES\PDF\BEQ-2.INP

Partial PDFs can be dynamically displayed each iteration of refinement or at the end of refinement. A dummy structure mechanism is used as follows:

```
str…
    ' main PDF phase
dummy_str
    phase_name "Al1 Al2 O1 O2 O3"
    pdf_partial_1 "Al1 Al2 O1 O2 O3"
    pdf_partial_2 "Al1 Al2 O1 O2 O3"
    pdf_partial_when 0 ' Only at end of refinement
dummy_str
    phase_name "V1 O4"
    pdf_partial_1 "V1 O4"
    pdf_partial_2 "V1 O4"
    pdf_partial_when = Mod(Cycle_Iter, 2);
```

pdf_partial_1 and pdf_partial_2 are site identifying strings which can include the '*' wild card character and the negation character '!'. pdf_partial_when determines when to do the partial pdf calculation; the default value is non-zero which means the calculation is performed each iteration. A value of zero results in he calculation being performed at the end of refinement. The BEQ-2.INP gives a GUI plot that looks like:



1.4..... Setting A-matrix elements that must-be-zero to zero

[approximate_A_check_for_must_be_zero #n]	**Example** XRD-CT\XRD-CT-1.INP

The approximate_A keyword uses the BFGS method to approximate the A-matrix. However, A-matrix elements that must-be-zero can still comprise non-zero values during the BFGS method. A_{ij} elements that must-be-zero include cases where parameter p_i and parameter p_j are from different xdd patterns.

The new keyword approximate_A_check_for_must_be_zero approximates the A-matrix using the BFGS method, but with elements that must-be-zero set to zero. This improves convergence in large problems where 1000s of xdds are being fitted with 100s of 1000s of independent parameters. Importantly, sparse matrix methods are invoked and only A-matrix elements that are non-zero are stored. In large problems this greatly reduces memory usage. Checking for zero A-matrix elements requires a modest amount of computational effort; to minimize this, the check is only performed up to the n^{th} iteration of a refinement cycle where n is the number defined after the approximate_A_check_for_must_be_zero keyword. After the n^{th} iteration, A_{ij} elements that must-be-zero are set to those that were zero at the n^{th} iteration. Example use is as follows:

```
approximate_A_check_must_be_zero = Cycle_Iter < 4;
```

1.5..... Stretching peaks

str...	**Examples**
[stretch_pks E]	STRETCH-PKS\STRETCH-1.INP

Refining 1000s of phases, where each has a peaks buffer that needs recalculation each iteration of the refinement, can be time consuming; as in XRT-CT refinements. In fact, many peak aberration parameters require the recalculation of the peaks buffer for each parameter derivative for each iteration of the refinement. The lor_fwhm and gauss_fwhm convolutions are two such parameters and typical usage is via the following macros:

```
CS_G(@, 100)
CS_L(@, 100)
```

When the values of the lor_fwhm and gauss_fwhm parameters are approximately known, then the shapes of the peaks can be approximated by stretching. For symmetric peaks the approximation is almost exact; asymmetric peaks, peaks with asymmetric convolutions, are not exact but if the values aren't too far off optimal values then the approximation can be good. The benefit of such an approximation is speed, where, using stretch_pks in the STRETCH-1.INP example speeds up refinement by a factor of 4.1. The usage of stretch_pks is as follows:

```
CS_L(100) ' not refined
CS_G(100) ' not refined
stretch_pks @ 1 min 0.001 max 10
```

The limits of a stretched peak, x1_s and x2_s, in terms of the unstretched limits x1 and x2, and the peak position Xo are:

x1_s = x0 – (Xo – x1) Get(stretch_pks)
x2_s = x0 + (x2 – Xo) Get(stretch_pks)

1.6..... Reusing objects in large refinements

lat_prms $name { ... } str_dets $name { ... } phase_dets $name { ... } use { ... }	**Examples** TEST_EXAMPLES\XRD-CT\XRD-CT-0.INP TEST_EXAMPLES\XRD-CT\XRD-CT-1.INP

The keywords lat_prms, str_dets and phase_dets can be used to define a set of lattice parameters, structural details and phase details that can be used multiple times within phases without recalculation of the corresponding item. The benefit is a reduction in memory usage and a speed up in refinement that is substantial when 100s of 1000s of phases are present. Similarly, derivatives of the common item are calculated once.

For a common phase with similar lattice parameters, then it is possible to use a commons set of hkls. Similarly, if the structure factors of the two phases are the same but the lattice parameters are different (but similar) then it is also possible to use a common set of structure factors. Absent the above keywords, the program automatically searches for common items in a global manner but with restrictions. For example, strs with hkls that are not identical cannot use a common str. However, defining the structure details in a str_dets object allows for a common structure even when the normal set of hkls generated would be vastly different.

XRD-CT-0.INP is a two str and two xdd example that highlights the use of the above keywords. It looks like:

```
' Change case_ to 0, 1
#prm case_ = 1;
iters 0
lam la 1 lo !lam 1 lg 0.3  ymin_on_ymax 0.001
str_dets s0 {
   space_group i41/amd:2
   site Zr x 0 y =3/4;      z =1/8;      occ Zr+4 1 beq !b1 1
   site Si x 0 y =1/4;      z =3/8;      occ Si   1 beq !b2 1
   site O  x 0 y !y1 0.066  z !z1 0.1951 occ O-2  1 beq !b3 2
}
lat_prms l0 { Tetragonal( 6, 4) }
lat_prms l1 { Tetragonal( 5, 7) }

prm !lor_ = Constant(0.1 Rad lam) / Cos(Th);
phase_dets pd0 { prm !cs0 140 min 10 max 500 lor_fwhm = lor_ / cs0; }
phase_dets pd1 { prm !cs1 100 min 10 max 500 lor_fwhm = lor_ / cs1; }

prm o10 0.01 min -0.1 max 0.1
prm o20 0.02 min -0.1 max 0.1
prm o11 0.03 min -0.1 max 0.1
prm o21 0.04 min -0.1 max 0.1
phase_dets ze0 { transform_X = o10 + o20 X + X; }
phase_dets ze1 { transform_X = o11 + o21 X + X; }

yobs_eqn aac1.xy = 1; min 30 max 60 del 0.01
   out_sfn4_ycalc = "xrd-ct-00.sfn4";
   bkg @ 100 -20 10
   #if case_ == 0;
      str scale @ 1 load use { l0 s0 pd0 ze0 }
   #elseif case_ == 1;
      str scale @ 1 load use { l0 s0 pd0 ze0 }
   #endif

yobs_eqn aac2.xy = 1; min 10 max 60 del 0.01
   out_sfn4_ycalc = "xrd-ct-01.sfn4";
   bkg @ 100 -20 10
   #if case_ == 0;
      str scale @ 1 load use { l1 s0 pd1 ze1 }
   #elseif case_ == 1;
      str scale @ 1 load use { l1 s0 pd0 ze1 }
   #endif
```

In the above, there are two strs and two xdds . In a real-world example this could extended to 100s of 1000s of xdds and strs resulting in an INP file comprising millions of lines. It is therefore efficient to define things once as is the case of lam. Modifying the preprocessor *case_ #prm* at the top of the file demonstrates capabilities. The *case_*=1 scenario is for the following:

- Two xdds each with one str
- The two strs use a common str_dets resulting in only one set of hkls being generated and on set of structure factor.
- The lattice parameters for the two strs are different and therefore two sets are used.
- The zero errors (transform_X) are different and therefore two sets used.
- The lor_fwhm peak shape convolutions (crystallite size) are different and therefore two sets are used.

*case_*=1 is an unrealistic example where the lattice parameters and x-axis of the two xdds are vastly different. The power of reusing object becomes apparent in a real-world sense where lattice parameters, amongst similar structures, are expected to be more similar. Important output from refinement for *case_*=1 is as follows:

```
Num data files: 2
Num hkl-sets/unique: 2 1
Num structure-factors-sets/unique: 2 1
Num m4_d2_inv unique: 1
Num peak buffers unique: 2
Num xo_ds unique: 2
Num bkg derivs unique: 2
Num transform_X/unique: 2 2
Num peak-shape-objects: 8
Num hkl_pk_dets/unique: 2 2
Num pk_sum_limits unique: 2

*** Warning: Lattice parameters not similar
    but using the same structure factors

   a 6 and 5
   b 6 and 5
   c 4 and 7
   al 90 and 90
   be 90 and 90
   ga 90 and 90
```

The unique items are shown in Red. Notice the warning which is due to the vastly different lattice parameters. *case_*=2 sets the peak shapes to be the same for the two phase and the output now looks like:

```
Num hkl-sets/unique: 2 1
Num peak buffers unique: 1
Num m4_d2_inv unique: 1
Num structure-factors-sets/unique: 2 1
```

Here we see one common peak buffer and thus only one is generated, and only derivatives for its parameters are calculated. Also seen is that one set of hkls is generated. The minimum/maximum x-axis values, used for the generation of the common hkls, corresponds to the minimum/maximum values of the the start_X/finish_X and extra_X_left/extra_X_right of all the common strs.

The example XRD-CT-1.INP refines on simulated data comprising 150,000 strs and 163,150 independent parameters. 20 iterations are completed in ~60s on an 8-core laptop. A few points to note when running XRD-CT-1.INP:

- Turn off animated fitting in the GUI, it cannot cope with 2000 xdd files and 150,000 strs.
- Run first with "#define CREATE_" to create the simulated data. The data files are created using the out_sfn4_ycalc keyword. This keyword outputs binary format files with a SFN4 extension. XY formats can also be outputted as well if desired.
- Do a first run with "*#define* SUBSET_" to see how things look (animated graphics can be turned on here).
- Then remove the *#define* and turn off animated graphics.

- 3.1 Gbytes of memory is used.

Output from the refinement looks like:

```
TOPAS-64 Version 8.38 (c) 1992-2020 Alan A. Coelho
   Maximum number of threads 8
Time   0.25, INP file pre-processed
approximate_A_check_must_be_zero On
Loading xyz's for fm-3m from file C:\w\sg\fm-3m.sg
Num hkls generated for C:\w\sg\fm-3m.sg 50
Loading xyz's for fm3m from file C:\w\sg\fm3m.sg
Num hkls generated for C:\w\sg\fm3m.sg 55
Loading xyz's for i41/amd:2 from file C:\w\sg\i41oamdq2.sg
Num hkls generated for C:\w\sg\i41oamdq2.sg 313
Num hkl-sets/unique: 150000 3
Num peak buffers unique: 3
Num independent parameters: 163150
Num data files: 2000
Num m4_d2_inv unique: 3
Num xo_ds unique: 3000
Num bkg derivs unique: 1
Num transform_X/unique: 150000 75
Num structure-factors-sets/unique: 150000 3
Num peak-shape-objects: 600000
Num stretch_pks/unique: 150000 3000
Num hkl_pk_dets/unique: 150000 3000
Num phase Ycalcs/unique (ignoring transform_X): 150000 3000
Num phase Ycalcs/unique derivs (ignoring transform_X): 150000 3000
Num pk_sum_limits unique: 3000
Num equiv posns for centrosymmetric fm-3m: 192
Num equiv posns for centrosymmetric fm3m: 192
Num equiv posns for centrosymmetric i41/amd:2: 32
  0  Time   5.37  Rwp   58.064    0.000 MC     0.00 0
  1  Time   7.12  Rwp   50.740   -7.325 MC     0.00 0
  2  Time  11.90  Rwp   45.643   -5.096 MC    11.10 3
  3  Time  15.77  Rwp   45.507   -0.137 MC   115.50 1
  4  Time  19.61  Rwp   43.351   -2.155 MC    30.34 1
approximate_A_check_must_be_zero: non-zero Aij elements now static
  5  Time  23.53  Rwp   25.599  -17.752 MC     8.31 1
  6  Time  26.62  Rwp   24.143   -1.457 MC    30.92 2
  7  Time  29.22  Rwp   14.654   -9.488 MC     8.05 1
  8  Time  32.26  Rwp   14.560   -0.094 MC   269.97 2
  9  Time  34.86  Rwp   14.480   -0.080 MC    68.26 1
 10  Time  37.48  Rwp   13.241   -1.239 MC    17.30 1
 11  Time  40.14  Rwp    5.025   -8.216 MC     4.67 1
 12  Time  43.23  Rwp    4.814   -0.211 MC    19.50 2
 13  Time  45.90  Rwp    4.097   -0.718 MC     5.18 1
 14  Time  48.98  Rwp    4.045   -0.051 MC    18.59 2
 15  Time  51.65  Rwp    3.783   -0.262 MC     4.85 1
 16  Time  54.30  Rwp    3.557   -0.226 MC     1.72 1
 17  Time  56.93  Rwp    3.512   -0.044 MC     3.51 1
 18  Time  59.62  Rwp    3.464   -0.048 MC    14.20 1
 19  Time  62.27  Rwp    3.374   -0.090 MC     3.29 1
--- 62.270 seconds ---
File C:\w\test_examples\xrd-ct\xrd-ct-1.out updated
    with parameters corresponding to best Rwp
```

Note the numbers in red. This is a large refinement that would not be possible without reusing objects and without the keyword approximate_A_check_must_be_zero. This refinement

cannot be tested against Version 7 as the number of hkls alone, 62,700,000, would exhaust much of memory.

Objects reused are:

- hkls
- lattice parameters
- Ycalc
- Peak buffers
- Structure factors
- th2_offset
- transform_X
- stretch_pks
- gauss_fwhm
- and many other common arrays such as (Sin(Th)/Lam)^2.
- derivatives for common refined parameters.

1.7..... 2θ point by point calculation of f0 and beq

Structure factors for powder diffraction data typically writes beq and the atomic scattering factor fo as a function of the Bragg angle $2\theta_o$. A more accurate description can be realized using the str dependent point_by_point_beq_fo_etc which writes the structure factor in terms of 2θ, or, for a particular reflection **h** we have:

$$F(\mathbf{h}, 2\theta) = \sum_s \left\{ \left(\sum_e e^{2\pi i\, \mathbf{h_e} .\mathbf{r}_{s,e}} \right) \left(\sum_a \left(f_{o,s,a}(2\theta) + f'_{s,a} + i f''_{s,a} \right) e^{-beq_{s,a}\, sin^2(\theta)/\lambda^2} O_{s,a} \right) \right\}$$

s corresponds to site *s*
e corresponds to the equivalent position of site *s*
a corresponds to atom *a*
$O_{s,a}$ corresponds to occupancy
beq corresponds to the beq parameter
f' and f'' corresponds to anomalous dispersion coefficients

This calculates fo and beq on a 2θ point-by- point basis rather than $2\theta_o$ or d-spacing. In routine Rietveld refinement the difference in structure factor values is small and difficult to detect. It can however be useful for analysing nanoparticles when extreme accuracy is required. The keyword only works with X-ray powder data and results in a slight increase in computational effort of ~5%. Reported structure factor values using the reserved parameter names of A01, B01, A11 and B11 are still written in terms of the Brang angle $2\theta_o$ and are therefore unchanged.

1.8..... To_Prm and using an equation to define a parameter name

Parameters names can be defined using equations by placing the % character before the equation-name. For example:

```
Create_XDDs(3)
prm i 1
for xdds {
    xdd_file = Concat("ceo2-", i, ".xdd");
        …
        str
            …
            site  Ce1                       occ Ce+4  1 beq %Concat("bCe", i); 0.2
            site  O1  x 0.25 y 0.25 z 0.25  occ O-2   1 beq %Concat("bO", i);  0.4
            existing_prm i += 1;
}
```

The above load three files called ceo2-1.xdd, ceo2-2.xdd and ceo2-3.xdd. Each file has a structure with two beq parameters created using the %Concat sequence. The names created are bCe1, bO1, bCe2, bO2, bCe3 and bO3, and these names can be used in equations as per normal. If the bCe_ parameters were to be equated to the bO_ parameters then the following could be used:

```
      site  Ce1                       occ Ce+4  1 beq %Concat("b1", i); 0.2
      site  O1  x 0.25 y 0.25 z 0.25  occ O-2   1 beq %Concat("b1", i); 0.2
```

or,

```
      site  Ce1                       occ Ce+4  1 beq %Concat("b1", i); 0.2
      site  O1  x 0.25 y 0.25 z 0.25  occ O-2   1 beq = To_Prm(Concat("b1", i));
```

Notice the use of the To_Prm function.

1.9 ..... Ingesting files into an INP file using #ingest

[*#ingest* $file]	

#ingest is a pre-processor command than copies a file into an INP file. The subsequent OUT file will also contain the ingested file. For example:

```
    xdd…
        str…
            #ingest common_str.txt
```

The output file will contain the ingested text with refined parameters updated. In other words, ingested files are treated as part of the original INP file. Ingested files can be nested. $file can be a function of macros.

1.10 ... #external_INP - using external INP format files

[*#external_INP* $file]	**Examples** TEST_EXAMPLES\EXTERNAL_INP\EXT_INP.INP

#external_INP is a pre-processor command than includes the file $file as part of the refinement without ingesting the text into the INP file. On refinement termination, the extension of $file is

changed to OUT and the contents of this OUT file is updated with refined parameter values. Example usage is as follows:

```
xdd…
    #external_INP instrument.inp
    #external_INP str.inp
```

An *#external_INP* file can contain further *#external_INP* commands. $file can be a function of macros. When running Launch mode from the GUI (TA.EXE), all *#external_INP* OUT files are renamed to INPs if the question on termination of refinement is answered in the affirmative.

1.11 … Defining hkls using use_hklm

hkls are automatically generated for str phases. This behaviour can be changed using the use_hklm keyword such that hkls become User-defined; for example:

```
str…
    load use_hklm {
      2   2   0  12
      2   2   2   8
      …
    }
```

1.12 … Defining local parameters using $

The $ character signals that a parameter is local to xdd scope or phase scope. The following two lines are similar but equivalent:

```
    xdd … local sc 0.01 min 1e-10 scale = sc;
    xdd … scale $sc 0.01
```

The advantage of using $ is that the default scale parameter attributes of min, max and del are retained. The $ character can also be used with the prm keyword resulting in the parameter being defined as local. The following two lines are equivalent:

```
    prm $cs 100
    local cs 100
```

Use of $ also simplifies the writing of macros when using “for { }” loops. For example, the following:

```
    for strs { CS_L(@, 100) }
```

expands to:

```
    for strs {
        prm m67cff550_1 100 min .3 max = Min(Val 2 + .3, 10000);
        lor_fwhm = 0.1 57.2957795130823 Lam / (Cos(Th) (m67cff550_1));
    }
```

Here, there's only one CS_L parameter, named m67cff550_1, for all strs within the loop. If on the other hand the intention was to have one unique CS_L for each str then the following can be used.

```
for strs { CS_L($cs, 100) }
```

which expands to:

```
for strs {
    prm $cs 100 min .3 max = Min(Val 2 + .3, 10000);
    lor_fwhm = 0.1 57.2957795130823 Lam / (Cos(Th) ($cs));
}
```

In the above each str has one unique cs parameter due to the use of the $ character. The problem with local parameters within the for loop, is that only one cs parameters is updated in the OUT file with the other cs parameter being lost. This situation can be remedied by using the keyword load_save_locals.

1.13 ... load_save_locals - automatically saving and loading parameters

[load_save_locals]	**Examples** TEST_EXAMPLES\LOAD-SAVE-LOCALS \LSL.INP

Parameters given unique names using the local parameters defined within "for {}" loops can be automatically saved and reloaded for subsequent refinements using the load_save_locals keyword. For example,

```
load_save_locals
xdd…
    str… phase_name p1
    str… phase_name p2
    for strs {
        site Ca1 x $ca1x 0.123 …
    }
```

In the above, there are two str's and two local x fractional coordinate parameters defined using the $ character. However, only one value is defined and thus only one value is saved to the OUT file on refinement termination. load_save_locals can be used to save both refined values to a file called INP_FILE.OUT_SL; notice the OUT_SL file extension. On rerunning the INP file, a check is made for the existence of a file called inp_file.sl. To therefore rerun the refinement with refined parameter values from the previous run then rename the INP_FILE.OUT_SL to INP_FILE.SL. This is what the GUI does. If the INP_FILE.SL exists, then on rerunning the INP file, both x factional coordinates are reloaded from the .SL file and assigned to their corresponding x parameter which is identified by the xdd file name and the phase name. If the file is a RAW file, then the range number is saved and used to identify the local parameters. Alternatively, the xdd dependent keyword xdd_tag can be used to identify the parameter instead of the xdd file name/range number. This is useful when xdd file names are the same as in the following:

```
XDD(..\ceo2) finish_X 50 …
XDD(..\ceo2) start_X  50 … ' Same file name as first xdd, need to use xdd_tag
prm i 0
for xdds {
    xdd_tag = Load_Eval(i);
    existing_prm i += 1;
    …
}
```

Phase names within a particular xdd needs to be unique. The Load_Eval function evaluates the parameter i when loading and places the value into the xdd dependent xdd_tag. A more complete example, LOAD-SAVE-LOCALS \LSL.INP, defines all refined values as local and is as follows:

```
load_save_locals
do_errors
XDD(..\ceo2) finish_X 50 str phase_name p1 str phase_name p2
XDD(..\ceo2) start_X  50 str phase_name p3 str phase_name p4
prm i 0
for xdds {
    xdd_tag = Load_Eval(Concat("tag", i)); ' Evaluate on load
    existing_prm i += 1;
   CuKa2(0.0001)
   Radius(173)
   LP_Factor(17)
   Full_Axial_Model(12, 20, 12, 5.1, $sl 5)
   Divergence(1)
   Slit_Width(0.1)
   Zero_Error($ze, 0)
   bkg $bkg  0 0 0 0 0
   One_on_X($onex, 0) ' This is a fit_obj phase which owns the its locals
   for strs {
      space_group FM3M
      scale $sc 0.001
      Cubic($a 5.4102)
      site  Ce1                      occ Ce+4  1 beq $b1  0.5
      site  O1  x 0.25 y 0.25 z 0.25  occ O-2   1 beq $b1  0.5
      CS_L($cs, 100)
   }
}
```

Notice that xdd_tag can optionally be a string. The OUT_SL file for the above is comma delimited and looks like:

```
"ze#tag0", 0.0102877916`_0.000480583326
"bkg_bkg0__#tag0", 17.434756`_1208.56088
"bkg_bkg1__#tag0", -15.6288801`_463.28484
"bkg_bkg2__#tag0", 12.6390723`_88.7458404
"bkg_bkg3__#tag0", 1.61512354`_16.6907443
"bkg_bkg4__#tag0", -3.11907939`_3.12933034
"onex#tag0", 1754.54902`_40988.0404
"sc#tag0#p1", 0.000715725375`_2.338e-06
"a#tag0#p1", 5.409464`_0.000061
"cs#tag0#p1", 231.533262`_1.80248407
"b1#tag0#p1", 0.12080`_0.03691
"sc#tag0#p2", 0.000715725375`_2.338e-06
"a#tag0#p2", 5.409464`_0.000061
"cs#tag0#p2", 231.533258`_1.79979947
```

```
"b1#tag0#p2", 0.12080`_0.03691
"ze#tag1", 0.0205123273`_0.000309444241
"bkg_bkg0__#tag1", 17.7487441`_136.299446
"bkg_bkg1__#tag1", 23.0641615`_68.4730749
"bkg_bkg2__#tag1", 6.76227976`_17.3604478
"bkg_bkg3__#tag1", -0.204165325`_4.17631833
"bkg_bkg4__#tag1", 1.86174398`_1.15703296
"onex#tag1", 2167.95208`_11401.1476
"sc#tag1#p3", 0.000736910097`_2.482e-06
"a#tag1#p3", 5.410297`_0.000013
"cs#tag1#p3", 193.208987`_0.979262806
"b1#tag1#p3", 0.25568`_0.00747
"sc#tag1#p4", 0.000736910097`_2.482e-06
"a#tag1#p4", 5.410297`_0.000013
"cs#tag1#p4", 193.208991`_0.97926241
"b1#tag1#p4", 0.25568`_0.00747
```

1.14 ... transform_x without recalculating patterns

[transform_x E]	**Examples** TRANSFORM_X\TPX.INP

The transform_x keyword stretches a calculated phase pattern to form a final phase calculated pattern. The following:

```
prm tpx 0 transform_x = X + tpx Sin(X Pi / 360);
```

is an approximation to:

```
prm tpx 0 th2_ffset = tpx Sin(X Pi/ 360);
```

This approximation is accurate when the change in transform_X is smooth and when its largest value is in the order of what is expected from XRD-CT data. For two common strs residing in different xdds, then if th2_offset were to be used then two th2_offsets would need to be defined and the formation of the summation of the peaks to the calculated pattern performed twice. transform_X on the other hand allows for the reuse of a common calculated str pattern. A further description is given in section **Error! Reference source not found.**.

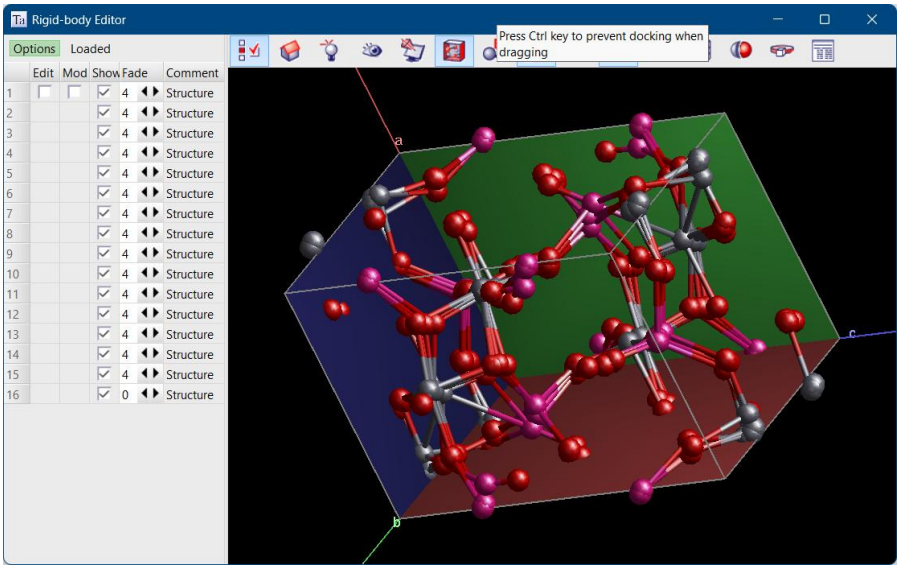
1.15 ... Tracking atomic movements graphically

[str...] [track_buffer !E] [site... track !E]	**Examples** TEST_EXAMPLES\ALVO4A.INP

Atomic movements can be tracked using the site dependent keyword track. For example, the following:

```
site Al1 … track = Mod(Cycle_Iter, 2) == 0;
```

will store the Al1 site position every second iteration. Doing this for every site in AlVO4 produces:



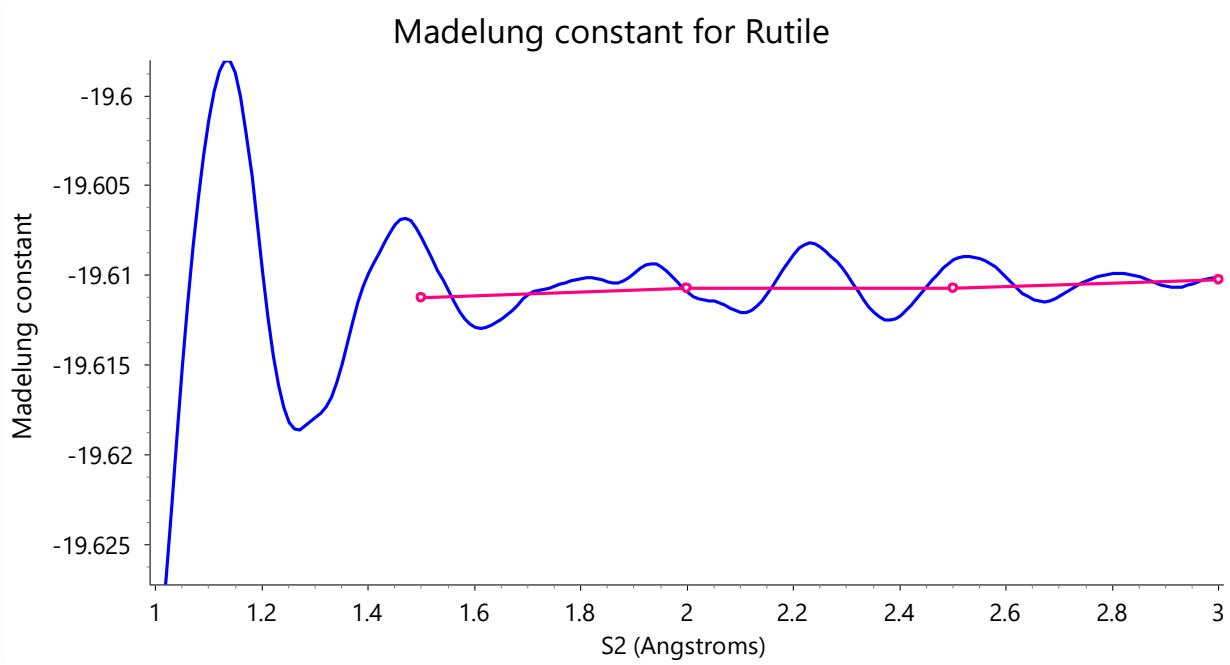
Saved positions are Faded with a Fade value of 4. A Fade value of 0 does nothing; a Fade value of 10 results in black atoms. The str dependent track_buffer keyword determines the number of previous atomic positions to keep; the default value is 10.

1.16 ... Energy Minimization

1.16.1 ..... Reporting on the Madelung constant

str...	**Examples**
[madelung #]	TEST_EXAMPLES\MADELUNG.INP

The madelung keyword reports on the Madelung constant of a structure (Madelung, 1918). It uses atomic charges defined at the occ keyword, see MADELUNG.INP. *#define show_GRS* in MADELUNG.INP creates an XY file with (S2-S1) of the GRS series set to 0.01. This is a small value that shows the behaviour of the GRS series which is as follows (blue line):



With the default values of S1 =1 and S2 = 1.5; the GRS series integrates between S1 and S2 to obtain an accurate value for the Madelung constant; this is seen in the first point of the Red line of the above plot. In energy minimization, the derivatives of the Madelung constant as a function of atomic coordinates constitutes the electrostatic force exerted on atoms.

1.16.2 ..... Reporting on the Coulomb potential at a site

site ... [co #]	

The site dependent keyword co reports on the Coulomb potential at a site. The sum of all co values equates to the Madelung constant. From observation, atoms of the same species seem to have similar co values in an ionic crystal. Note, both the co and madelung keywords are independent of the grs_interaction keyword.

1.16.3 ..... Enhancements to the grs_interaction

site ... [g !N q E s E] [repulsion_refine] [grs_interaction [qi !E qj !E] $s1 $s2 c !E]... [no_coulomb] [penalty = Get(grs_lp_rep);] [penalty = Get(grs_lp_refine);]	**Examples** TEST_EXAMPLES\ ALVO4-GRS-AUTO.INP (not new) GRS-ALVO4\SOLVE-1.INP GRS-ALVO4\REP-1.INP GRS-ALVO4\REP-2.INP

The site dependent keyword g reports on the difference in value between the sum of all grs_interactions with the site included, and the site excluded. In other words, it reports on the site's contribution to all grs_interactions.

When repulsion_refine is defined, then all grs_interactions are placed in a "repulsion refine" mode. In this mode, grs_interactions return the sum of the derivatives squared of the grs_interactions, with respect to the atomic coordinates, or, in pseudo code:

$$\text{Sum}(d\text{grs\_interaction}/df_i, i)^2$$

where f_i corresponds to the x, y and z coordinates of the sites associated with the grs_interaction. In this manner, a refinement will adjust repulsion parameters such that the derivatives of the grs_interactions with respect to independent repulsion parameters are a minimum.

Repulsion parameters include qi, qj, q, s and any other parameters defined in the grs_interaction equation. The new site dependent keyword, s, scales the equation part of grs_interactions. This simplifies the setting up of grs_interactions; consider the following:

```
grs_interaction qi = 3; qj = -2; Al* O* p1 = B1 / R^7; penalty = p1;
grs_interaction qi = 5; qj = -2; V*  O* p2 = B2 / R^7; penalty = p2;
grs_interaction qi = 3; qj =  5; Al* V* p3 = B3 / R^7; penalty = p3;
```

Here, there are three parameters B1, B2 and B3. In the repulsion_refine mode, fractional coordinates are not refined. However, their derivatives with respect to the grs_interaction

equations are expensive and are required. The site dependent s parameters can be used to avoid this recalculation as follows:

```
site Al … q  3 s s1 1
site V …  q  5 s s2 1
site O …  q -2 s s3 1
repulsion_refine
grs_interaction * * c = 1/R^9;
penalty = c;
```

Note the reformulation where three grs_interactions become one. Here the program examines the equation, 1/R^9 in this case, and, if independent of refined parameters, the program stores the 1/R^9 values for use in the calculation of derivatives of the grs_intercations equations with respect to repulsion refined parameters. This results in a large speed up in computation. The three parameters s1, s2 and s3 are related to the B1, B2 and B3 values as follows:

B1 = s1 s3
B2 = s2 s3
B3 = s1 s2

These parameters are related to the minimum distance R_o between two isolated atoms *i* and *j*, and for the case of opposite q charges, as follows:

$$U_{ij} = q_i\, q_j / R + s_i\, s_j / R^n$$

Setting the derivative to zero:

$$dU_{ij}(R=R_o)/dR = 0$$

we get:

$$R_o = [\, (n-1)\, s_i\, s_j / (q_i\, q_j)\,]^{1/(n-1)}$$

1.16.4 ..... Including lattice parameter in grs_interaction(s)

The default is to not include lattice parameters when repulsion_refine is defined. To include the minimization of derivatives of the grs_interactions with respect to the lattice parameters, the following can be used:

```
penalty = Get(grs_lp_rep); : 0
```

For normal refinement (repulsion_refine not defined), lattice parameters, flagged for refinement, are included in the derivatives of grs_interactions if the following is included:

```
penalty = Get(grs_lp_refine); : 0
```

1.16.5 ..... Ignoring the Coulomb part of the grs_intercation

The Coulomb part of the grs_interaction can be ignored using the no_coulomb keyword. This is useful for materials that are not wholly ionic. Various version of the Lennard Jones potential, for example, can be implemented; consider a potential U of:

$U = A / R^6 + B/R^{12}$

To efficiency calculates this U, then two grs_intercations can be used:

```
site... q  1 s @ 1
site... q -1 s @ 1
grs_interaction … = 1 /R^4; no_coulomb
grs_interaction … = 1 /R^9; no_coulomb
```

Here, 1/R^4 and 1/R^9 values are stored in lookup tables which are calculated once at the start of refinement. This potential is used in describing the partly ionic structure of $AlVO_4$, see GRS-ALVO4\REP-2.INP.

The grs_interaction equation can also be set to zero. This may be useful when looking at dipole properties of a molecule where the centre of the electron cloud is at a different position from the nucleus, for example:

```
site Al1       x x1 # y y1 # z z1 # …
site Al1_shift x = dx1 x1; y = dy1 + y1; z = dz1 + z1; …
grs_interaction … Al1 = 0;                      ' No repulsion equation
grs_interaction … Al1_shift = 1 /R^9; no_coulomb ' No Coulomb potential
```

1.16.6 ..... _rem attribute - Removing/inserting parameters from refinement

The _rem parameter attribute is an equation that is evaluated at the start of a refinement iteration (note: attribute equations cannot be named). If non-zero, the associated parameter is removed from refinement for the duration of the iteration. The parameter can be reinstated in subsequent iterations if _rem evaluates to zero; for example, to reinstate the parameter after convergence and into a new Cycle, the following could be used:

```
prm a 1 _rem = Mod(Cycle, 2);
```

1.16.7 ..... Using ok_to_continue and _rem

In version 7, the ALVO4-GRS-AUTO.INP test example was the fastest way of solving the $AlVO_4$ structure. In that example, the scattering power of the Al and V sites are allowed to refine within the scattering power range of Al+3 and V+5. An alternative to refining on the Al+3 occupancies, is to fix the occupancies for a certain number of Cycles and then change the occupancies on the Al+3 site without actually including them in the refinement. This is accomplished using the new keywords of ok_to_continue, q, s and the new _rem parameter attribute; it works as follows (see GRS-ALVO4\SOLVE-1.INP for details):

```
macro qal { 3 } ' Charge of Al+3
macro qv  { 5 } ' Charge of V+5
macro exp { 9 } ' Repulsion exponent

macro ro_al { 1.65 } ' Ro values from bond distances
macro ro_v  { 1.5 }
macro ro_oo { 2.4 }

' Change Ro values to s values
```

```
prm !so  = Sqrt((4 ro_oo^(exp-1)) / exp); :  22.1184
prm !sal = ((ro_al^(exp-1)) 6 / exp   ) / so; :  1.65587133
prm !sv  = ((ro_v^(exp-1)) 10 / exp) / so; :  1.28746033

macro S_ { If(Get(q) == qal, sal, sv) }
macro OCC { If(Get(q) == qal, 1, 1.85) }
macro VV { rand_xyz 1 } ' Randomize sites at start of cycle
macro VQ { _rem 1 val_on_continue = If(Mod(Cycle,10), If(Rand(0,1)<0.5,qal,qv), Val); }
prm q1 qal VQ
prm q2 qal VQ
prm q3 qal VQ
prm q4 qv  VQ
prm q5 qv  VQ
prm q6 = 3 qal + 3 qv - q1 - q2 - q3 - q4 - q5;

' Ensure scattering power equals 3*Al sites plus 3*V sites
ok_to_continue = Or(q6 == qal, q6 == qv);

Grs_(*, *, exp, 0) ' grs_interaction penalty

site Al   x @  0.0  y @  0.9  z @  0.8 occ Al+3 = OCC; q = q1; s = S_; VV
site Al   x @  0.1  y @  0.0  z @  0.9 occ Al+3 = OCC; q = q2; s = S_; VV
site Al   x @  0.2  y @  0.1  z @  0.0 occ Al+3 = OCC; q = q3; s = S_; VV
site Al   x @  0.3  y @  0.2  z @  0.1 occ Al+3 = OCC; q = q4; s = S_; VV
site Al   x @  0.4  y @  0.3  z @  0.2 occ Al+3 = OCC; q = q5; s = S_; VV
site Al   x @  0.5  y @  0.4  z @  0.3 occ Al+3 = OCC; q = q6; s = S_; VV

site O1   x @  0.6  y @  0.5  z @  0.4 occ O-2 1 q -2 s = so; VV
site O2   x @  0.7  y @  0.6  z @  0.5 occ O-2 1 q -2 s = so; VV
site O3   x @  0.8  y @  0.7  z @  0.6 occ O-2 1 q -2 s = so; VV
site O4   x @  0.9  y @  0.8  z @  0.7 occ O-2 1 q -2 s = so; VV
site O5   x @  0.0  y @  0.9  z @  0.8 occ O-2 1 q -2 s = so; VV
site O6   x @  0.1  y @  0.0  z @  0.9 occ O-2 1 q -2 s = so; VV
site O7   x @  0.2  y @  0.1  z @  0.0 occ O-2 1 q -2 s = so; VV
site O8   x @  0.3  y @  0.2  z @  0.1 occ O-2 1 q -2 s = so; VV
site O9   x @  0.4  y @  0.3  z @  0.2 occ O-2 1 q -2 s = so; VV
site O10  x @  0.5  y @  0.4  z @  0.3 occ O-2 1 q -2 s = so; VV
site O11  x @  0.6  y @  0.5  z @  0.4 occ O-2 1 q -2 s = so; VV
site O12  x @  0.7  y @  0.6  z @  0.5 occ O-2 1 q -2 s = so; VV
```

The above will change the scattering power on the Al* sites every 10th Cycle as defined in the VQ macro. Note, there's only one grs_interaction and the Grs_ macro looks like:

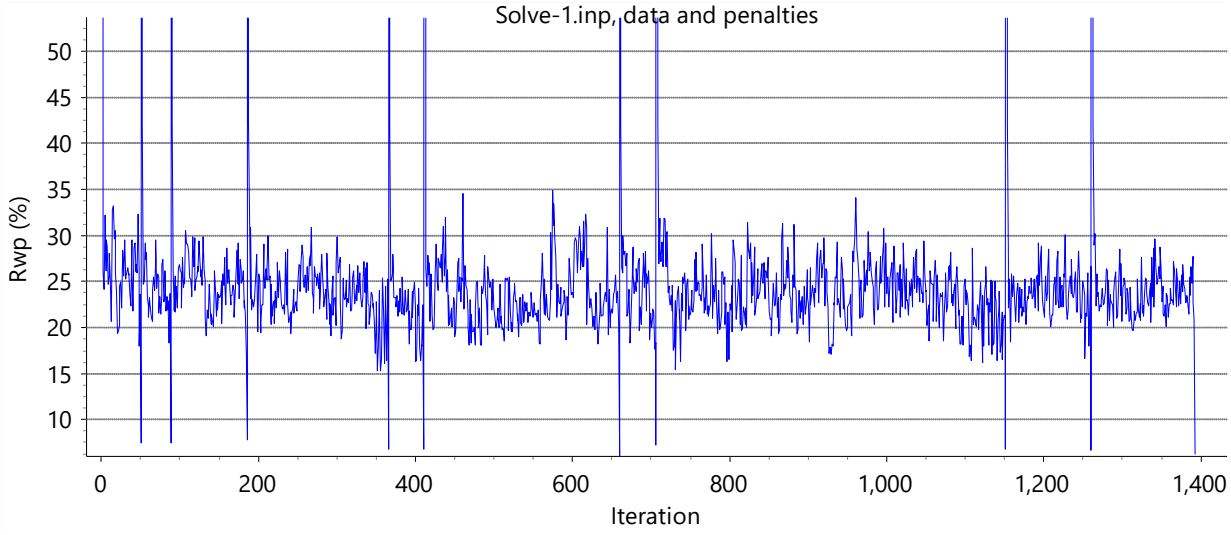
```
    macro Grs_(s1, s2, & n, v)
        {
            grs_interaction s1 s2 #m_unique c =
                If (R < rsm,
                    ((-n rsm^(-2 - n)/2) R^2 + rsm^(-n) + n/(2 rsm^n)),
                    1 / R^n
                );
            penalty = c; : v
        }
```

SOLVE-1.INP operates in three modes which can be chosen by the two control parameters (in Red) in the INP text at the top of the file and is as follows:

```
continue_after_convergence
#prm penalties_only_start_at_Rietveld_positions = 1;
#if penalties_only_start_at_Rietveld_positions;
    only_penalties
    verbose 1
    temperature 0.5 use_best_values
#else
    #prm solve_using_real_data_and_penalties = 1;
    #prm solve_using_penalties_only = solve_using_real_data_and_penalties == 0;
    verbose -1
    num_runs 10  ' Solve structure 10 times, change to 1 to see solution

    #if solve_using_real_data_and_penalties;
        ' Minimum energy at 5%
        temperature = If(Mod(Cycle, 200), 0.7, 10);
        iters = If(And(Cycle_Iter > 2, Get(r_wp) < 8), 0, 1000000000);
    #endif
    #if solve_using_penalties_only;
        ' Minimum energy at -423.5
        only_penalties
        temperature = If(Mod(Cycle, 200), Rand(0.35, 0.7), 10);
        iters = If(And(Cycle_Iter > 2, Get(r_wp) < -423), 0, 1e9);
    #endif
#endif
```

Running SOLVE-1.INP with *penalties_only_start_at_Rietveld_positions*=1 refines on the atomic coordinated with only_penalties defined. It also displaces the atomic positions by an amount of rand_xyz*temperature, or, 0.5 Å in a random direction at the start of each cycle. As can be seen whilst running, the structure returns to the Rietveld refined values after each cycle.

Running SOLVE-1.INP with *solve_using_real_data_and_penalties*=1 solves the structure 10 times and the Rwp plot looks like:



This is similar to ALVO4-GRS-AUTO.INP which refines on occupancies. ok_to_continue is evaluated at the start of each iteration. If it evaluates to zero, then val_on_continue of its independent parameters are executed. The process is repeated until all ok_to_continue(s) evaluates to non-zero. Note, more than one ok_to_continue can be defined.

1.16.8 ..... Energy minimization-only resulting in the observed structure of AlVO4

Running SOLVE-1.INP with *solve_using_real_data_and_penalties*=0, achieves a minimum energy configuration that matches the Rietveld refined structure. only_penalties are refined; lattice parameters are not included. Even though $AlVO_4$ is partly ionic, the maximum atomic displacement at the energy minimum compared to the Rietveld refined positions is relatively small at ~0.22 Å with an average movement of ~0.14 Å. In other words, the energy minimization "pseudo-solved" the structure from a crude atomic interaction model.

Including lattice parameters as refinable parameters results in non-sensical atomic coordinates which means that the atomic interaction model is inadequate in a physical sense.

1.16.9 ..... Determining repulsion parameters for AlVO4

REP-1.INP performs three types of operations/refinements as seen by the self-explanatory control statements at the top of the file of:

```
#prm determine_repulsion_parameters = 0;
#prm test_rep_prms = 0;
#prm bond_length_differences = 0;
```

Setting *determine_repulsion_parameters*=1 fixes the atomic coordinates to Rietveld refined values and then minimizes $dU_{ij}/df_i=0$ for $AlVO_4$ by varying three s repulsion parameters of *sal*, *sv* and *so* where:

$$U_{ij} = q_i q_j / R + s_i s_j / R^9$$

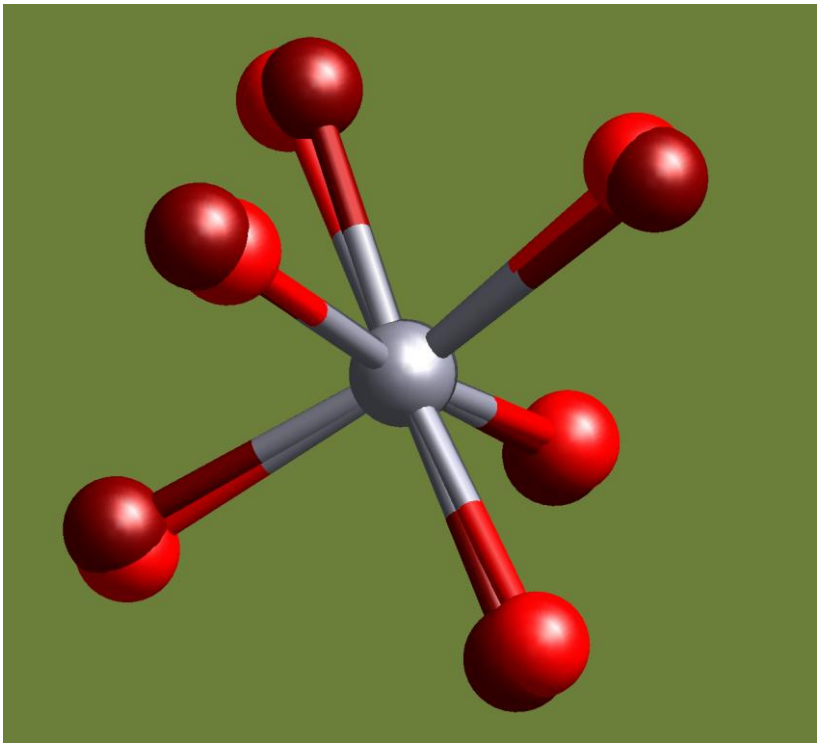
As seen in REP-1.INP, the sum of the derivatives squared of dU/df_i (where f_i is a fractional atomic coordinate) do not refine to zero. This is seen in the lines:

```
Grs_(*, *, 9, 0.465098047`)
penalty = Get(grs_lp_rep); :  2.6397897`
```

Also, seen in REP-1.INP is that the R_o values (distance between two isolated atoms) seem too large as in:

```
prm !ro_alo = ( (exp-1) Abs(sal so qal qo) )^(1/(exp-1)); :  2.18729482
prm !ro_vo  = ( (exp-1) Abs(sv  so qv  qo) )^(1/(exp-1)); :  2.4673052
prm !ro_oo  = ( (exp-1) Abs(so  so qo  qo) )^(1/(exp-1)); :  2.73559269
```

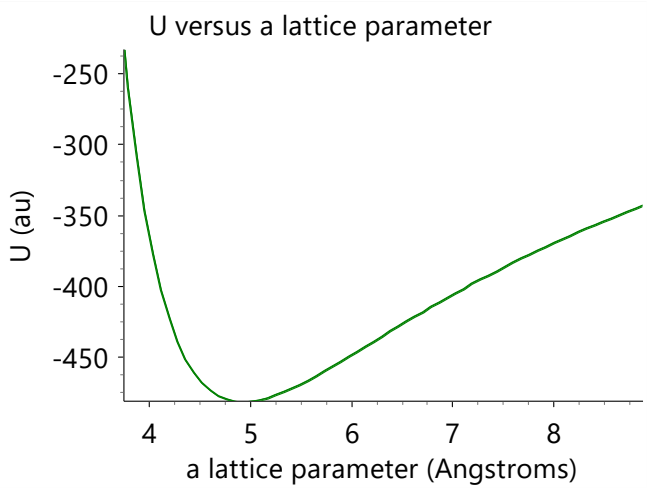
Performing another refinement with the three determined repulsion parameters *sal*, *sv* and *so* fixed, and instead refining on the atomic coordinates (*test_rep_prms*=1) results in a structure with average atomic movements of 0.14 Å from the Rietveld coordinates. The movement can be seen in the following Al octahedron (lighter atoms are the Rietveld determined positions):



Setting *test_rep_prms*=1 and *output_U_vs_a*=1 executes the INP code of:

```
#if And(output_U_vs_a, test_rep_prms);
   verbose 1
   num_runs 100
   iters 0
   a = Ramp_Run_Number(6.54131-3, 6.54131+3, Get(num_runs));
   out a.xy append
      Out(Get(a))
      Out_String(" ")
      Out(Get(non_fit, r_wp))
      Out_String("\n")
#else
   a  6.54131
#endif
```

This produces the XY file of:



Here we see that the observed *a* lattice parameter of 6.54131 Å is far from the minimum; this was evident from the non-zero value for Get(*grs_lp_rep*) as seen above. Note the use of *Get(non_fit, r_wp)* instead of *Get(r_wp)*; the former gets the global Rwp and the latter the xdd dependent Rwp. Use of only_penalties does not update xdd dependent Rwp(s); hence the need to Get the *global r_wp*.

Lattice parameters were not refined in performing the *test_rep_prms*=1 operation; they could have been with the inclusion of the line:

```
penalty = Get(grs_lp_refine); : 0
```

The refinement in such a case would have produced very incorrect results as indicated by the U versus *a* plot above. This demonstrates that a simple Coulomb sum and 1/R^9 repulsion term does not fully describe $AlVO_4$ and that another model is needed.

1.16.10..... A non-ionic model for AlVO4

Instead of using the Coulomb sum, a 1/R^4 term was used for atoms of opposite charge (Al-O and V-O) and a 1/R^9 for like charges, or,

$$U_{ij} = A_{ij} / R^4 + B_{ij}/R^9$$

This U choice was a guess and there may well be more physically meaningful models available. The following however does highlight the ability to quickly model such cases. The REP-2.INP test example uses this potential and it has three operational/refinement modes:

```
#prm repulsion_refine = 0;          ' set to 0 or non-zero
#prm bond_length_differences = 0;   ' set to 0 or non-zero
#prm test_repulsion_prms = repulsion_refine == 0;
```

Refining with *repulsion_refine*=1 results in a low value for *grs_lp_rep* and for grs_interactions:

```
penalty = Get(grs_lp_rep); : 0.000423118927`
Grs_(Al*,  O*, ea,  -a_alo, 0.000824217622`)
Grs_(V*,   O*, ea,  -a_vo,  0.00103650359`)
Grs_(O*,   O*, ea,  a_oo,   0.000721564661`)
Grs_(Al*, Al*, ea,  a_alal, 0.000102652961`)
Grs_(Al*,  V*, ea,  a_alv,  0.000417591887`)
Grs_(V*,   V*, ea,  a_vv,   0.000314938926`)
Grs_(Al*,  O*, er,  b_alo,  0.000824217622)
Grs_(V*,   O*, er,  b_vo,   0.00103650359)
Grs_(O*,   O*, er,  b_oo,   0.000721564661)
Grs_(Al*, Al*, er,  b_alal, 0.000102652961)
Grs_(Al*,  V*, er,  b_alv,  0.000417591887)
Grs_(V*,   V*, er,  b_vv,   0.000314938926)
```

These are low values compared to those obtained for REP-1.INP and it indicates near zero values for $(\mathrm{dgrs\_interaction}/\mathrm{df_i})^2$ where f_i is a fractional atomic coordinate or lattice parameter. The difference in lattice parameters between the observed values from Rietveld refinement and the energy minimization of REP-2.INP is:

```
Δa  = 0.353377, Δb  = 0.387755,  Δc = 0.501125
```

```
    Δal = -0.92222, Δbe = -0.32539, Δga = -0.77862
```

The maximum bond length difference is 0.18 Å with an average difference of 0.08 Å.

1.17 ... Molecular dynamics (MD)

molecular_dynamics md_time_step !E (default = 0.002) md_time !E md_scale !E (default = 1) New parameter attributes: _md_k !E (default = 1) _mass !E (default = 1) _md_force !E (default = 0)	**Examples** TEST_EXAMPLES\GRS-ALVO4\ MD-1.INP MD-2.INP MD-3.INP MD-4.INP GRS-0.INP

1.17.1 ..... Molecular dynamics in a general manner

Defining molecular_dynamics (MD) places the program in a non-refinement mode where parameters of any type can be updated in a time dependent manner. The Verlet (1967) algorithm is used for updating parameters. In the present implementation, parameters that are not typically associated with molecular dynamics can be updated in a MD manner. This is accomplished with the use of the parameter attributes of _md_k and _md_mass.

Molecular dynamics is basically the steepest decent method of refinement but with new parameters values accepted regardless of the change in the objective function (Rwp in the case of TOPAS), or, relating this to the Newtonian equations of motion for iteration *k* and parameter *p*, we have:

$$\text{Force}(k) = m\, a(k) = d\text{Rwp}/dp$$

$$\text{Velocity}(k+1) = \text{Velocity}(k) + (d\text{Rwp}(k)/dp) / m$$

In the Verlet algorithm, velocity is not considered explicitly and instead p is updated as follows:

$$p(k+1) = 2\,p(k) - p(k-1) + a(k)\,t^2$$

$$= 2\,p(k) - p(k-1) + (1/m)\,(d\text{Rwp}(k)/dp)\,t^2$$

where *t* is the time step of the molecular dynamics. The mass *m* is set using _mass. To introduce flexibility, the present implementation allows modifications of *p*(k+1) as follows:

$$p(k+1) = (p(k) - p(k-1) + (\text{\_md\_k} / \text{\_mass})\,(d\text{Rwp}(k)/dp)\,t^2)\,\text{md\_scale} + p(k)$$

This equates to the Verlet algorithm with the default value of 1 for _md_k, _mass and _md_scale. md_scale is a means of increasing or decreasing atomic movements (increasing or decreasing temperature).

1.17.2 ..... Molecular dynamics for atoms

In the absence of the _mass attribute, mass is determined from the masses found in the ISOTOPES.TXT file for site occupancies, as defined by the occ keyword, and weighted by the occ values. In the absence of the _md_k attribute, md_k is determined for x, y and z coordinate parameters as follows:

md_k for x = ax
md_k for y = by
md_k for z = cz
where ax = 1
bx = Cos(Get(ga) Deg)
by = Sin(Get(ga) Deg)
cx = Cos(Get(be) Deg)
cy = (Cos(Get(al) Deg) - cx bx) / by
cz = Sqrt(1.0 - cx^2 - cy^2)

The following two sites are therefore equivalent:

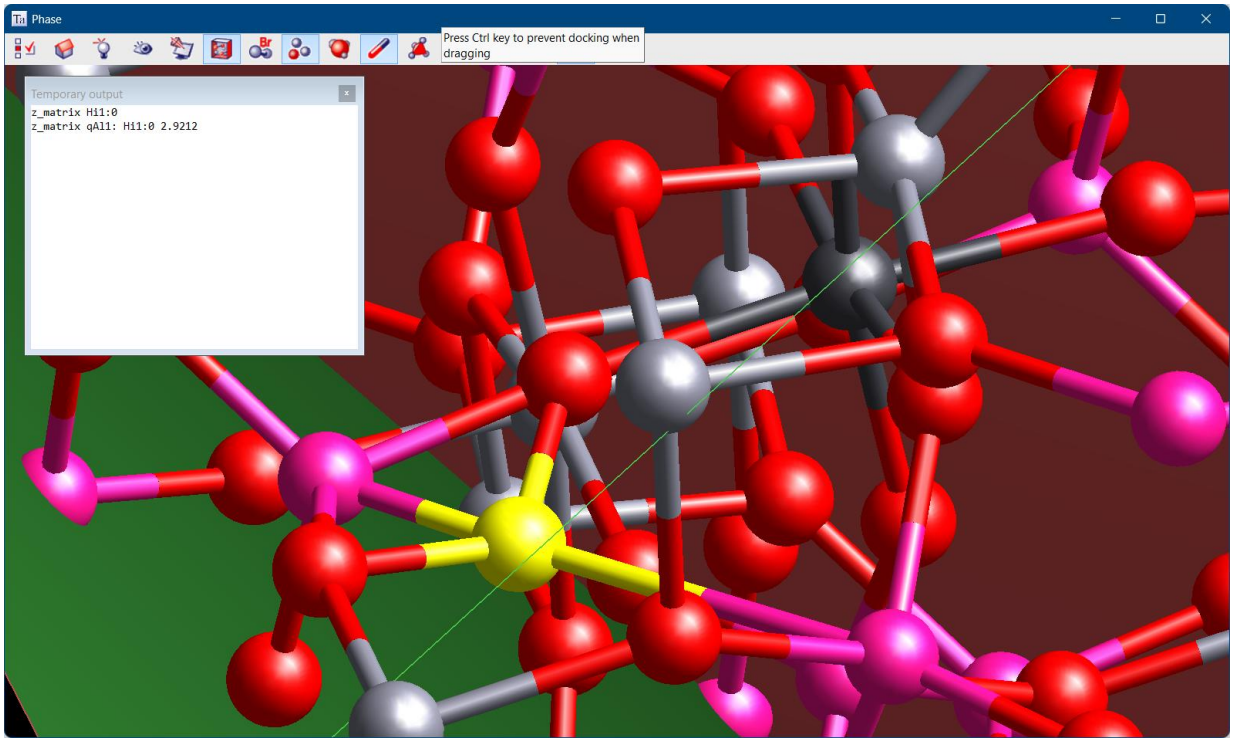
```
    site Al
        x @ # _mass = 26.981; _md_k = ax;
        y @ # _mass = 26.981; _md_k = by;
        z @ # _mass = 26.981; _md_k = cz;
        occ Al+3 1
' and
    site Al x @ # y @ # z @ # occ Al+3 1
```

The use of _md_k corrects the forces in case of non-orthogonal lattice parameters. grs_interaction can be used to calculate the forces for molecular dynamics; this is demonstrated in MD-1.INP. The display of the Rwp plot in the Fit dialog can take a lot of processing for long MD runs; not displaying the plot can speed up the simulation; and imilarly for the OpenGL 3D graphics.

MD-1.INP operates in the P-1 space group; this can be changed to P1 by outputting the fractional coordinates in P1 as follows:

```
    p1_fractional_to_file aac.txt
        in_str_format
        in_cart 0
        na 2 nb 2 nc 2
```

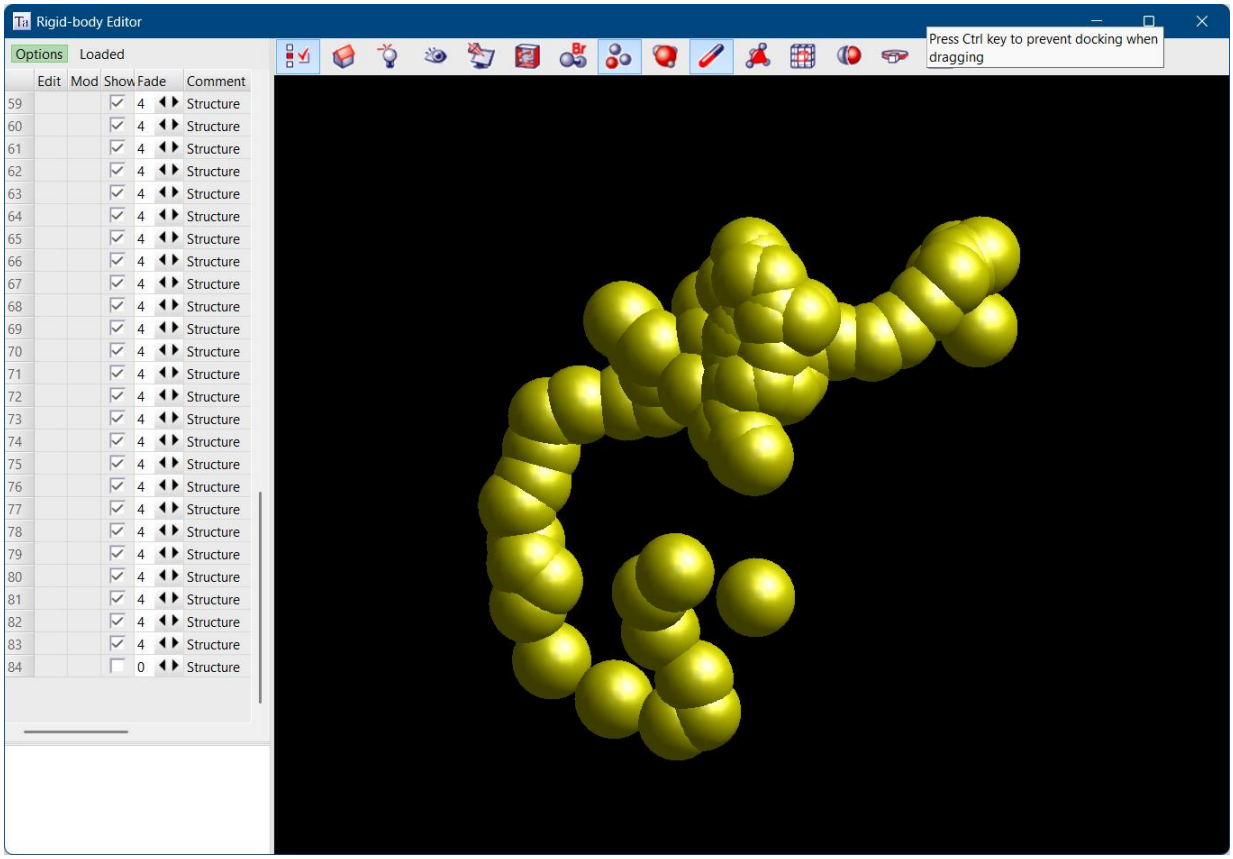
Here, a 2x2x2 unit cell is outputted in P1 to the AAC.TXT file. MD-2.INP describes such a unit cell comprising 288 atoms. One of the AL1 atoms is offset, and running the MD simulation results in the atom returning to its lowest energy configuration position. This return to the optimal position is due to the small offset of 2.92 Å. The following shows the starting configuration:



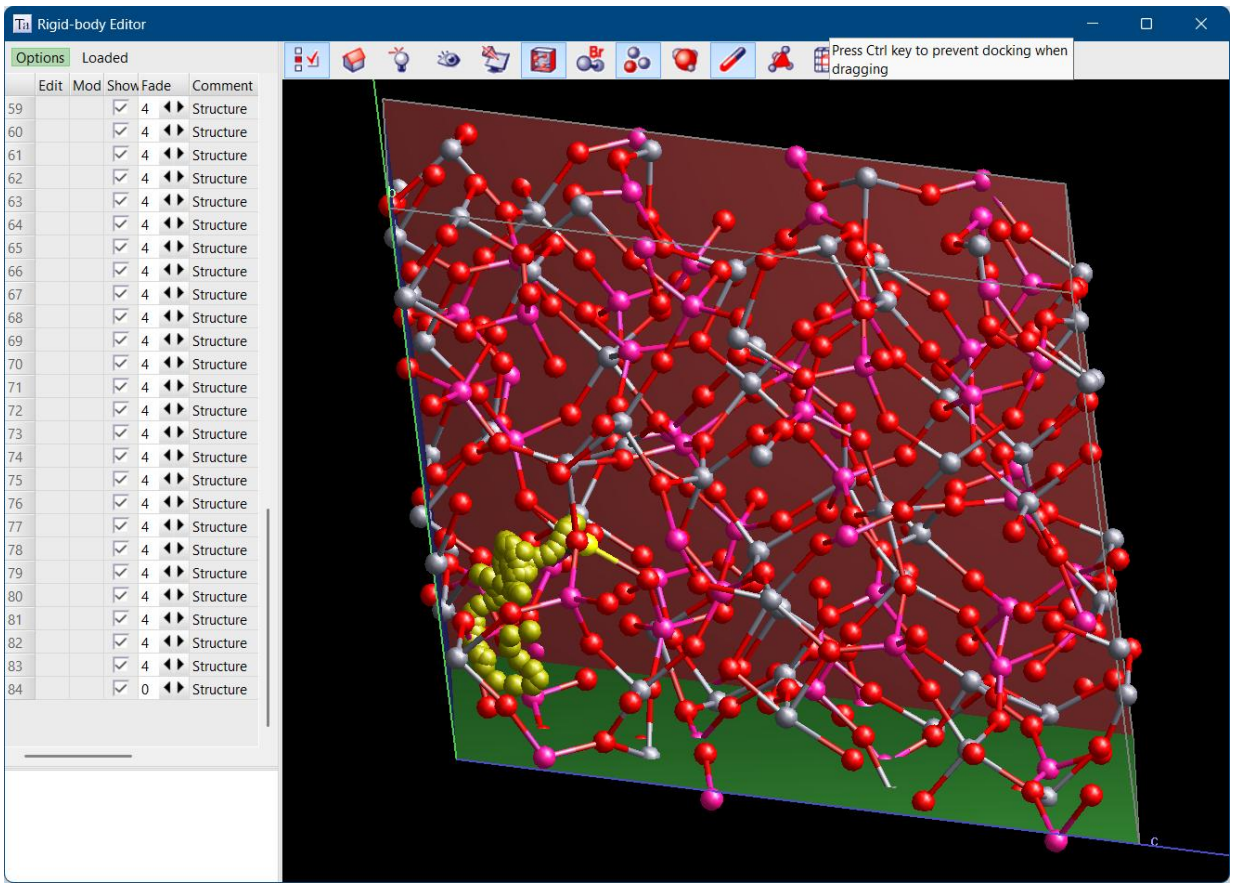
where the yellow atom is the Al1 atom's offset, and the dark grey atom is the original position of the Al1 atom. It is informative to watch the yellow atom migrate to the dark grey atom. The INP text that produces the coloured Al1 sites is:

```
track_buffer 100
site qAl1 x   0.37342 y   0.34930 z   0.20369  occ Al+3  1 ' original posn
site Hi1  x @ 0.2     y @ 0.2     z @ 0.1  occ Al+3  1 track = Mod(Cycle_Iter, 20) == 0;
```

The colours for the Al1 and Hi1sites is seen in the ATOM_COLORS.DEF file; this file can be edited for the purpose of changing atom colours. The original atom qAl1 does not take part in the only_penalties MD simulation as it is absent from the grs_interactions. The path of the Al1 atom looks like:



Note the last display has been disabled (item 84); it comprises all 288 atoms in the cell. Including line 84 produces:



MD-3.INP moves the Al1 atom 4 Å from the original position and in a random direction; the pertinent INP text is:

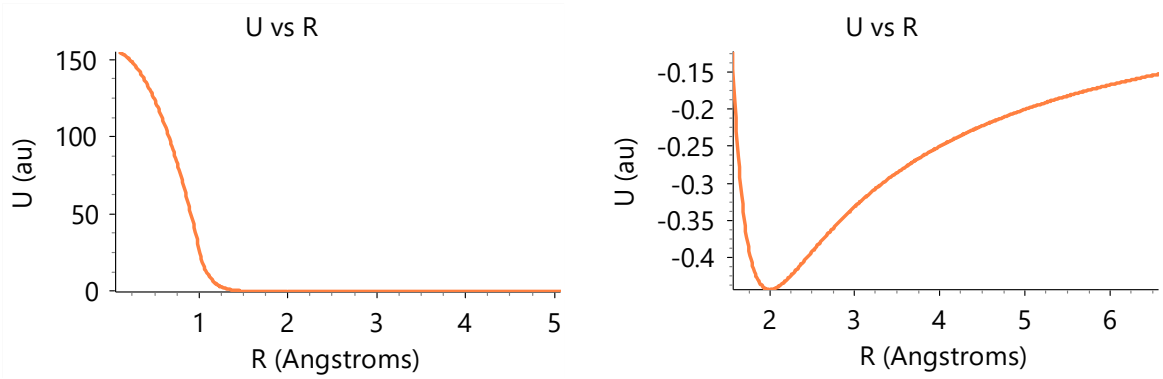
```
temperature 1
md_scale = If(Cycle_Iter < 2, 0.1, 1);
site Hi1 x @  0.37342 y @  0.34930  z @ 0.20369  occ Al+3 1 rand_xyz 4
```

Clicking on the Break icon; ie.

executes rand_xyz. Often the energy of the system (which is kept constant) is too great and the MD goes chaotic. This behaviour can be damped using md_scale as seen above. Also, repulsion terms such as $1/R^9$ can be very large when R is small; such small bond distances are unrealistic and modifying U_{ij} to avoid large values is beneficial. In the present work the equation used for the grs_interaction, rewritten in terms of a yobs_eqn, is given in GRS-0.INP, or,

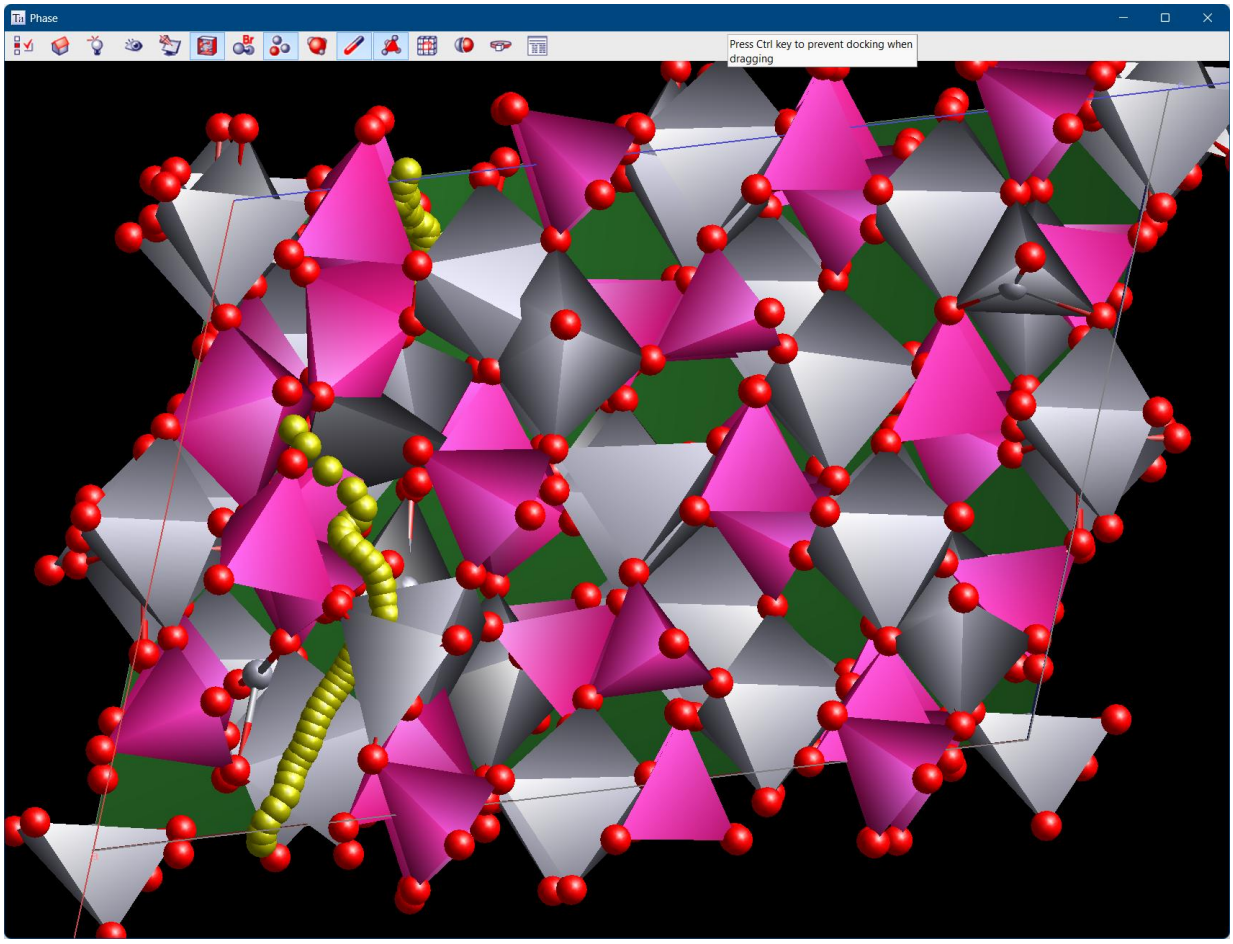
```
macro & n { 9 }
macro & q { -1 }
macro & ro { 2 }    ' x-axis value at the minimum
macro & rsm { 1 }
yobs_eqn aac.xy =
   If (X < rsm,
        (Abs( q ) / n) (ro^(n-1)) ( (-n rsm^(-2 - n)/2) X^2 + rsm^(-n) + n/(2 rsm^n)),
        (Abs( q ) / n) (ro^(n-1)) / X^n + q / X
   );
    min 0.1 max 7 del 0.01
```

Note, the rsm value of 1. For R < rsm, U is modified such that large values are not encountered. The following shows two views of the same yobs_eqn plot:



1.17.3 ..... Applying a force on atoms

The _md_force attribute can be used to apply a force to atoms. The MD simulation in such a case maintains energy conservation by adjusting the kinetic energy of the system. For the case of $AlVO_4$ and for the crude potential used; it is interesting that for a force along the a-axis on an Al+3 atom, the structural integrity is maintained as seen below (see MD-4.INP):



The structure, however, loses its integrity for a similar force along the c-axis.

1.18 ... Cross correlation function

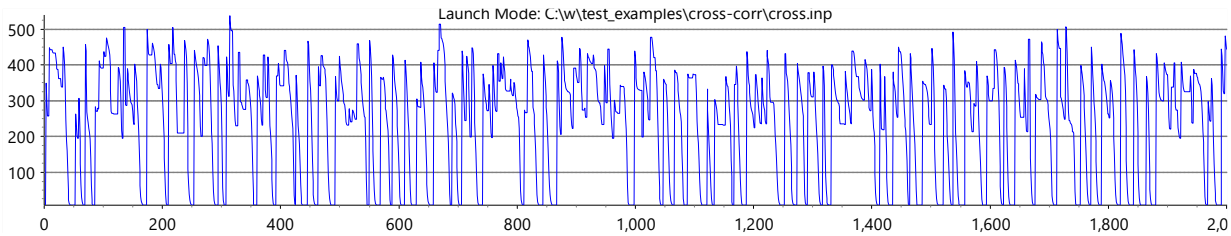
[cross_corr $name #value cross_corr_s !E	**Examples** CROSS-CORR\CROSS.INP

cross_corr calculates the cross-correlation function for a triangle of x-axis width cross_corr_s. cross_corr_s can be an equation that can be a function of *Cycle_Iter* which allows for changing the width of the triangle in situ. $name is a name that can be given to the function and #value is the value of the cross-correlation function. $name can be used in the chi2 keyword for obtaining lattice parameters. However, as can be seen in the example CROSS.INP, using normal refinement with a triangle convolution is much faster than using the cross-correlation function. CROSS.INP is an informative example and it looks like:

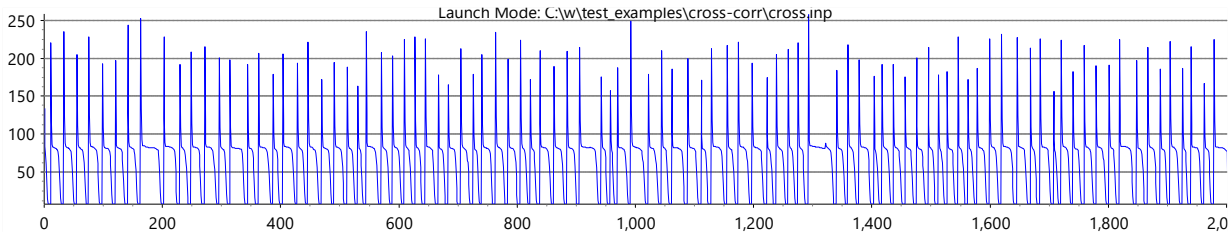
```
#prm USE_CROSS_CORRELATION = 1; ' Set to zero to see normal refinement
#prm INCLUDE_HATS = 1;          ' This is for normal refinement
macro DEL_ { Rand(-1, 1) 0.5 }  ' Change in lattice parameters at the start of a Cycle
macro AA { }

continue_after_convergence
verbose 1
iters 2000

RAW(..\pbso4)
   rebin_with_dx_of 0.01
```

```
CuKa5(0.0001)
LP_Factor(17)
Radius(173)
Full_Axial_Model(10, 10, 10, !sol  3.77616`, !sol  3.77616`)
Divergence(1)
Slit_Width(0.2)
bkg AA -792.524948 767.974856 -305.050785 121.658117 -45.020282 18.2136589
One_on_X(AA, 22265.9137`)

ZE(AA,-0.0110740988)

finish_X 60
extra_X_right 10

#if (USE_CROSS_CORRELATION)
   cross_corr corr 0
      cross_corr_s 3
   chi2 = -Ln(corr); : 0
   macro SCALE_ {  }
#else
   ' Normal refinement
   #if (INCLUDE_HATS)
      hat @ 1 val_on_continue 2 max 2 num_hats 2
   #endif
   macro SCALE_ { @ }
#endif

STR(P_b_n_m) ' PbSO4
   space_group P_b_n_m
   macro LP_(v) { v val_on_continue = v + DEL_; min = v - 3; max = v + 3;
   a  @  LP_(6.962377)
   b  @  LP_(8.483133)
   c  @  LP_(5.400478)
   site Pb x AA 0.16717 y  AA 0.18778 z 0.25       occ Pb+2 1 beq AA 1.47495
   site S  x AA 0.18429 y  AA 0.43563 z 0.75       occ S    1 beq AA 0.85254
   site O1 x AA 0.09441 y  AA 0.59667 z 0.75       occ O-2  1 beq AA 1.05681
   site O2 x AA 0.03611 y  AA 0.31151 z 0.75       occ O-2  1 beq AA 1.63474
   site O3 x AA 0.31549 y  AA 0.42069 z AA 0.97553 occ O-2  1 beq AA 1.49181

   CS_L(AA, 274.77)
   Strain_L(AA, 0.035898)
   scale SCALE_  0.000335087199
```

Running cross.inp with USE_CROSS_CORRELATION set to 1 gives an Rwp plot of:



Running cross.inp with USE_CROSS_CORRELATION set to 0 (normal refinement) gives an Rwp plot of:


Launch Mode: C:\w\test_examples\cross-corr\cross.inp
250
200
150
100
50
0
200
400
600
800
1,000
1,200
1,400
1,600
1,800
2,0

2. ..MISCELLANEOUS

2.1..... User defined phase colour, line width and point size (_clp)

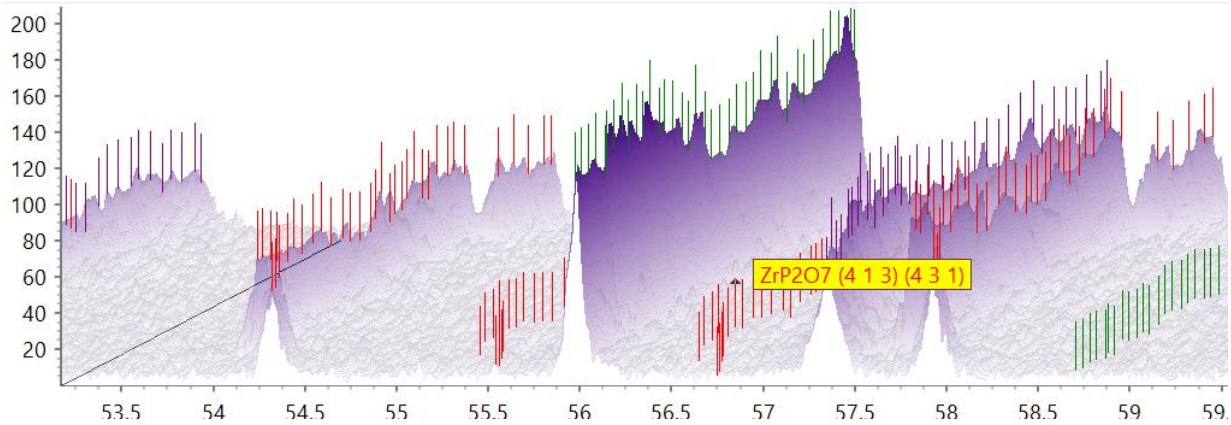
_clp #red #green #blue #line_width #point_size	**Examples** TEST_EXAMPLES\ZRO2.INP

The colour, line width and data point size of phase plots can be entered in INP files using the phase or bkg dependent keyword _clp. The first three numbers correspond to red, green and blue colour weightings with value ranging between 0 and 1. #line_wdith and #point_size can be between 0 and 15. The file COLOURS.INC contains standard colours. Example usage is as follows:

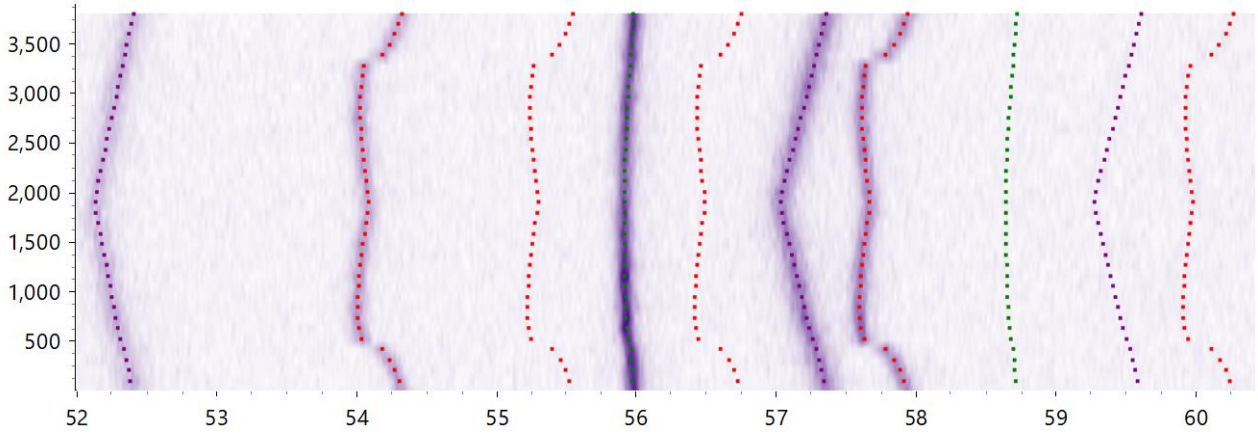
```
#include colours.inc
bkg …
    _clp 0.5 0.5 0.5 3 2 ‘ Grey with a line width of 3 and a data point size of 2
str…
    _clp 0.2 0.2 1 1 0   ‘ Blueish with a line width of 1 and a data point size of 0
fit_obj !fs  = 1000 X;
    Plot_Fit_Obj(fs, Some_bkg)
           _clp Blue 2 2 ‘ Blue colour from colours.inc
```

2.2..... Display hkl ticks on Surface plots

hkl ticks with z-axis height can be displayed on surface plots as seen in the following for \TEST_EXAMPLES\JE-PARA\D8_02999_35_ANNOTATE_04.INP:

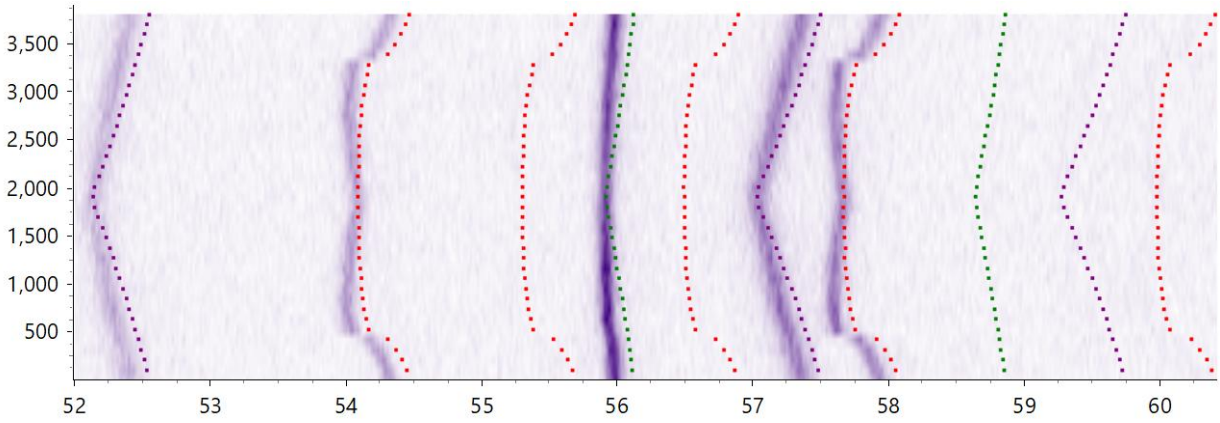


And in PlanView:



2.3..... hkl ticks are now corrected for zero errors

The display of hkl ticks in Surface or 1D plots are now corrected for th2_offset or transform_X. The corrected PlanView plot shown above, when uncorrected, looks like:

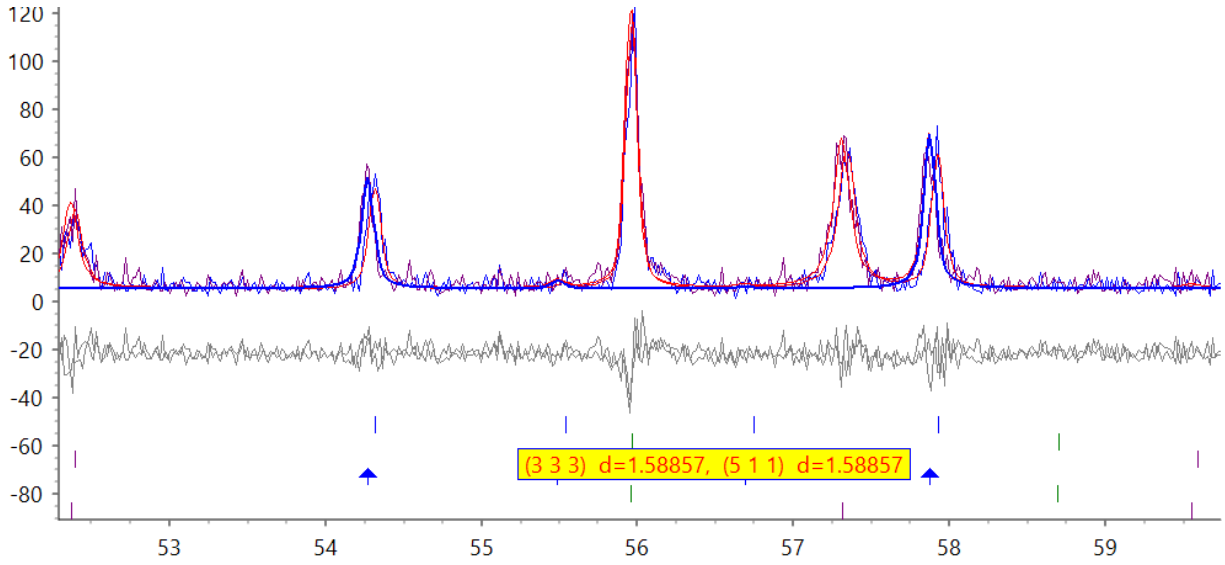


2.4..... Highlighting/displaying phases and hkl tick marks

Highlighting a phase in a multi-phase pattern or patterns, can be performed in a many ways:

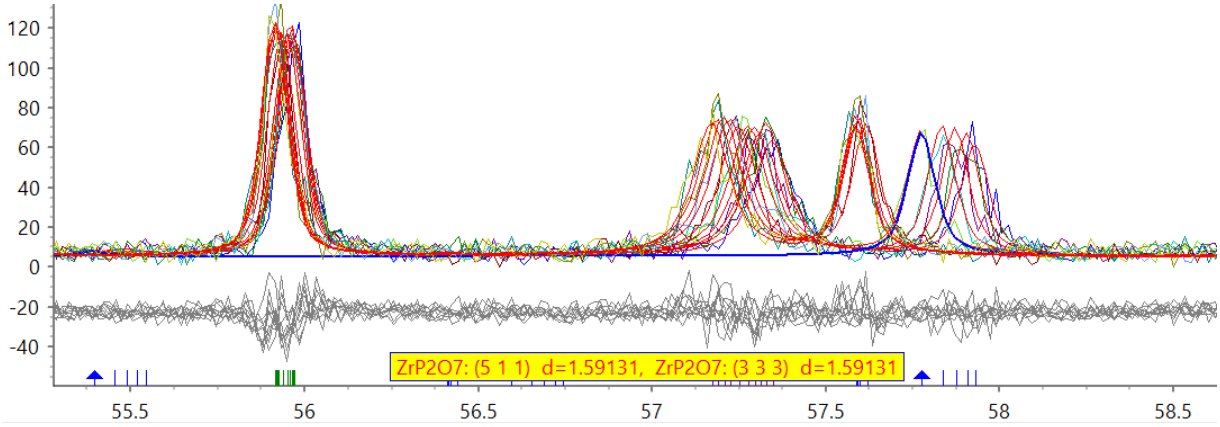
- Displaying phase names on the right of the plot window and moving the mouse over the phase name.

- Clicking on the row of the hkl tick marks. This displays the associated phase and tick marks highlighted, for example:

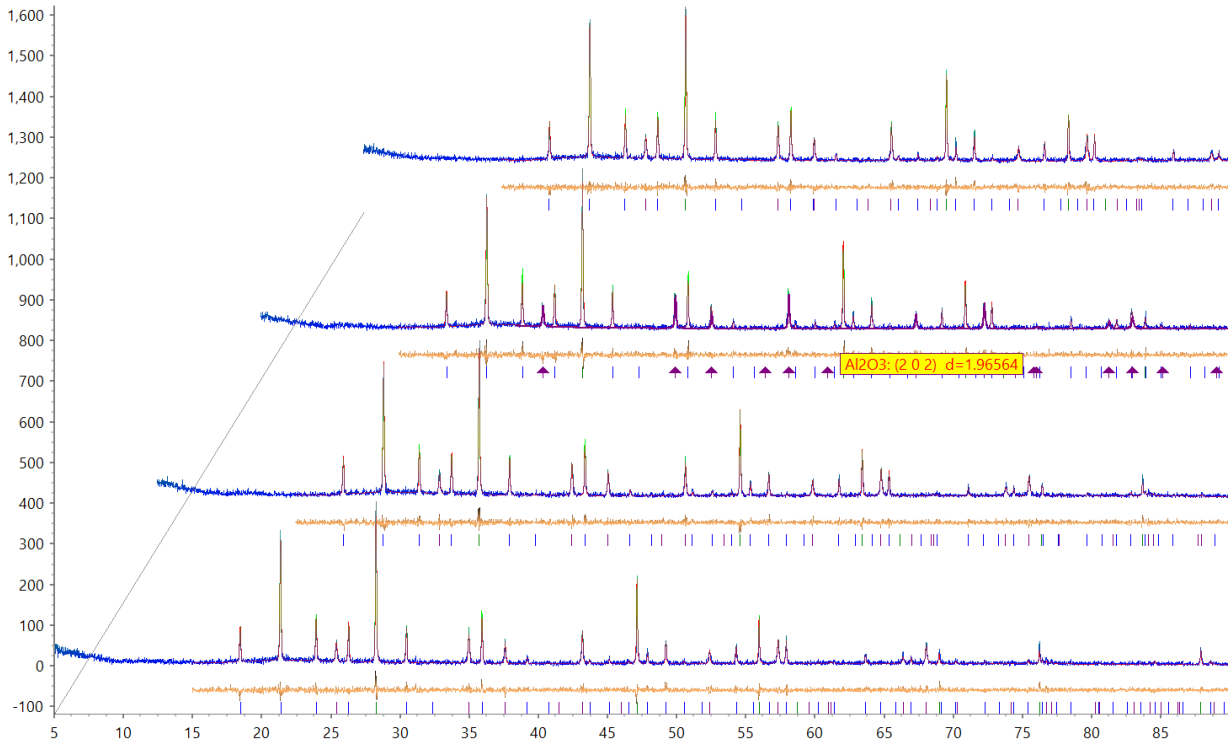


If individual phases are displayed using  , then moving the mouse over the pattern highlights the pattern as well as ticks marks associated with the phase.

If there are too many tick rows, then the program displays all phase ticks from all patterns in one row. Moving the mouse close to a tick mark, displays tick mark information on the screen as well as displaying the associated range name on the status bar. Additionally clicking the left mouse button on a phase tick mark, highlights all tick marks associated with that phase, and additionally displays the phase pattern itself highlighted; for example:



hkl tick marks are also now displayed in 2D-Offset mode as seem in the following:



In 2D-offset mode, ticks marks from a particular pattern are placed on a common tick mark row. Clicking on an individual tick mark, highlights all ticks marks from the corresponding phase and highlights and displays the corresponding phase pattern. As in the non-2D-offset mode, a phase pattern and its associated tick marks are highlighted when the mouse is close to the phase pattern.

2.5..... gui_text keyword now ignored by the kernel

For the commercial version of TOPAS, the gui_text keyword allows for INP format text to be used in GUI mode refinement. Previously, use of gui_text within INP files in Launch mode threw an exception. Version 8 does not throw an exception; instead, the keyword is ignored but with the INP text within the gui_text block included in the refinement. The means that INP files are GUI mode compliant and moving between GUI and Launch mode becomes a smooth operation. For example, the following INP file can be run in both Launch and GUI modes without modification:

```
XDD(ceo2)
   CuKa5(0.0001)
   Full_Axial_Model(12, 20, 12, 5.1, sl 5)
   Radius(173)
   LP_Factor(17)
   Divergence(1)
   Slit_Width(0.1)
   bkg  @ 0 0 0 0 0
   Zero_Error(@, 0)
   gui_text {
      prm b0 0
      prm b1 0
      fit_obj = b0 + b1 (X2-X1) / 2;
   }
   str
```

```
space_group F_M_3_M
scale @ 0.001
Cubic(@ 5.410)
site Ce1                        occ Ce+4 1 beq 1
site O1  x 0.25 y 0.25 z 0.25   occ O-2  1 beq 1
CS_L(@, 100)
MVW(0,0,0)
```

2.6..... Outputting special characters

The characters ,(){}[] can be outputted to text files by enclosing them in double quotation marks. For example:

```
out aac.txt
    Out_String("a,b}c(d")
```

To output a single apostrophe or double quote character, the escape sequences of %A and %B can be used respectively, for example, the following:

```
do_errors
prm b -0.61427_0.01048
out aac.txt
    out_record
        out_fmt "%V g[h%Bh"
        out_eqn = b;
    out_record
        out_fmt "\n%s%A (brack}et"
        out_eqn = "abc";
```

produces in the AAC.TXT file:

```
-0.614(10) g[h"h
abc' (brack}et
```

The escape sequences themselves can be outputted by using two separate out_records. For example, to output %A use:

```
Out_String("%")
Out_String("A")
```

2.7..... Iterating over internal data-tree nodes using 'for'

'for' can be used to iterate over all nodes of the internal data tree. For example, to iterate over the site_recs node the following can be used:

```
for site_recs {
    beq @ 1
}
```

2.8 ..... Command prompt output during INP file loading using print

The keyword print is executed during the loading of INP files; it is useful for determining when an item is loaded for debugging purposes. For example:

```
print("Executed during the loading of INP files")
#prm a 1.234
print(#out a)
```

2.9 ..... Sorting output by columns using _sort_dec or _sort_inc

Columns can be sorted in ascending or descending order using _sort_inc or _sort_dec respectively. For example, the following can be used to sort by d-spacing in descending order and then by I_no_scale_pks in ascending order:

```
xdd...
   str...
      phase_out aac.txt
         out_record
            out_fmt "  d = %9.5f"
            out_eqn = D_spacing; _sort_dec 1
         out_record
            out_fmt "  I_no_scale_pks = %9.5f"
            out_eqn = I_no_scale_pks; _sort_inc 2
         out_record
            out_fmt "  2Th = %9.5f\n"
            out_eqn = 2 Rad Th;
```

2.10 ... Creating many xdds at once using new and xdd_file

The new keyword can be used to create many xdds at once, for example:

```
macro Create_XDDs(n)
    {
        move_to xdds
        move_to xdd_recs
        new(xdd, n)
    }
```

See section 1.1 for usage of the macro Create_XDDs.

2.11 ... seed, #seed_eqn, seed-tc.txt, seed-tb.txt, Rand

If using the command line TC.EXE, then seed increments the number in the file SEED-TC.TXT to seed the random number generator and then saves the incremented value back to SEED-TC.TXT file. If using the GUI version of the program TA.EXE, then the number in the file SEED-TB.TXT is used. *#seed* is similar except it is executed at the pre-processor stage of loading INP files. If a number occurs after either seed or *#seed* then the random number generator is seeded with that number.

#seed_eqn seeds the random number generator at the preprocessor stage with the equation value; here are examples:

```
#seed_eqn seed_1 = Rand(0,32767);
prm value_fed_to_random_number_generator = #out seed_1;
#seed_eqn seed_2 = Run_Number;
```

The rand function of c++ returns integers between 0 and 32767. To increase the dynamic range, the function Rand returns a random number between two floating point numbers r1 and r2, in c-code, as follows:

```
double Rand(double r1, double r2)
    {
        const double c0 = 1.0 / double(RAND_MAX);
        const double c1 = 1.0 / (1.0 + double(RAND_MAX));
        return (rand() + rand() * c0) * c1 * (r2 – r1) + r1;
    }
```

2.12 ... Getting the number of items in a #list using #list_n

During the pre-processor phase of loading INP files, #list_n, a new pre-processor command, returns the number of items in a #list; for example:

```
#list Files { file1.xdd file2.xdd file3.xdd }
Create_XDDs(#list_n Files)
```

2.13 ... out_prm_vals_on_end

In addition to out_prm_vals_per_iteration and out_prm_vals_on_convergence, the new keyword, out_prm_vals_on_end, allows for output only at the end of refinement, for example:

```
str…
    prm wtp = Get(weight_percent);
    out_prm_vals_on_end aac2.txt append
       out_prm_vals_filter wtp
       out_prm_vals_dependents_filter wtp
```

The following example shows how to add items to the out_prm_vals_on_end file.

```
    #list Files { file1.xy file2.xy }
    num_runs #list_n Files
    do_errors
    out_prm_vals_on_end results.txt #if (Run_Number > 0) append #endif
    xdd Files(Run_Number)
        str…
        out results.txt append
            load out_record out_fmt out_eqn {
                "%d " = Run_Number;
                " %s  " = Files(Run_Number);
                }
```

The above will output the following into the file RESULTS.TXT.

```
Cycle Iter Rwp second_soll1FCF42E6640_ Err bkg1FCF3E90F18 Err ...
    0      7  7.026593e+00  7.437574e+00  3.511447e-02  1.208843e+01 ...
    0  file1.xy
Cycle Iter Rwp second_soll1FCF42E6640_ Err bkg1FCF3E90F18 Err ...
    0      7  7.026593e+00  7.437574e+00  3.511447e-02  1.208843e+01 ...
    1  file2.xy
```

2.14 ... Modify_peak now works with no_th_dependence

modify_peak can now be used with no_th_dependence. When no_th_dependence is defined then Get(current_peak_x) returns the x-axis of the point being calculated; when no_th_dependence not defined then Get(current_peak_x) returns the wavelength of the point being calculated.

2.15 ... Not saving extrapolated peaks when doing intensity derivatives

str... [dont_save_extrapolated_pks]	

The process of adding peaks to a calculated profile from the peaks buffer can be computationally intensive. This process occurs many times during a refinement iteration when, for example, calculating the derivatives of a site fractional atomic coordinate. An in-between step is therefore performed where interpolated peak data is stored. The memory requirements for the interpolated data can be large and in cases where memory is an issue then the keyword dont_save_extrapolated_pks can be used.

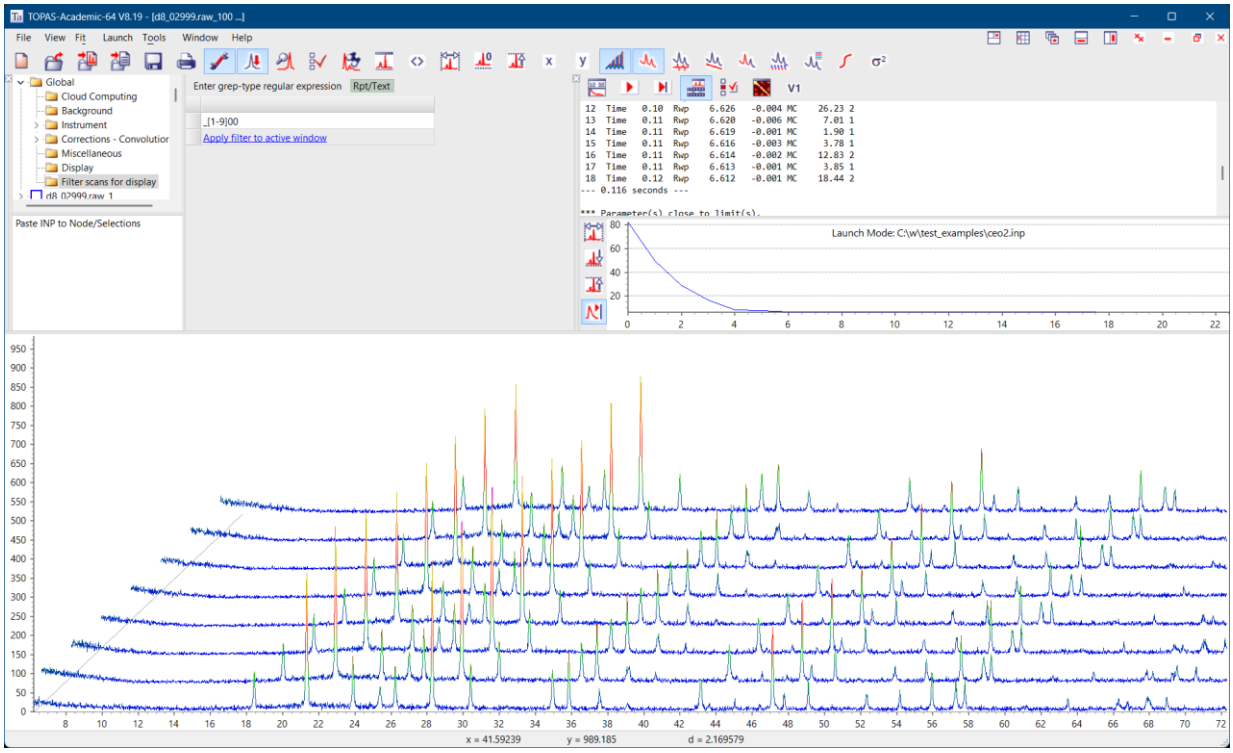
2.16 ... Atomic f0 values from a table

User defined atomic f0 values can be read from a file where the format comprises d-spacing and f0 value. A user_y object is used to hold the f0 values, the following demonstrates its usage:

```
xdd…
   MoKa2(0.001)
   user_y !C_f0_table C_f0_table.xy ' x-axis are the D_spacings
   str
     load f0_f1_f11_atom f0 f1 f11 { C = C_f0_table; 0 0 }
     Cubic(8.61)
     space_group "201"
     site C1 x 0 y 0 z 0 occ C 1 beq 1
```

2.17 ... Selecting files for display using grep regular expressions

Grep regular expressions can be used to simplify the selection of scans for display; this is useful when there are many patterns loaded. Grep can be accessed through the “Global/Filter scans for display” option in the TreeView pane as seen in the following:



In the above, every 100th scan is displayed using the regular expression of "_[1-9]00".

2.18 ... Applying lp_search to TOF data

lp_search cannot be directly applied to TOF data. However, it is relatively easy to convert the TOF data to 2Th data where lp_search can be used. The examples TEST_EXAMPLES\TOF\TOF-TO-Q.INP is an example that converts the data to 2Th and then applies lp_search. The conversion is as follows:

I(Q) = Intensity(TOF) dTOF/dQ

Q = 2 Pi / d

d = 2 Pi / Q

TOF = t0 + t1 d + t2 d^2 = t0 + t1 2 Pi / Q + t2 (2 Pi)^2 / Q^2

dTOF/dd = t1 + 2 t2 d;

dd/dQ = -2 Pi / Q^2;

dTOF/dQ = dTOF/dd dd/dQ

If t0 = t2 = 0 we get:

dTOF/dQ = -t1 2 Pi / Q^2 = -t1 D_spacing^2 / (2 Pi)

Or in INP format we have:

```
xdd TOF-DATA.XY
   x_calculation_step = Yobs_dx_at(Xo) .5;prm !t0 0
   prm !t1 6171.89377
   prm !t2 0
   prm !d = X / t1;
   prm !Q = 2 Pi / d;
   prm !dtof_dd = t1 + 2 t2 d;
   prm !dd_dQ = -2 Pi / Q^2;
   prm !dtof_dQ = dtof_dd dd_dQ;
   xdd_out TOF-to_Q.xy load out_record out_fmt out_eqn
      {
          " %11.6f  " = 2 Pi / d;
          " %11.6f\n" = Yobs Abs(dtof_dQ);
      }
```

2.19 ... Correction for dispersion using modify_peak_eqn

Example: TEST_EXAMPLES\DISPERSION\DISP.INP

The shape of the emission profile changes with 2θ due to dispersion such that:

I(lam) dlam = I(th) dth

or,

I(th) = I(lam) dlam_dth

Differentiating Bragg's with respect to θ we have:

dlam_dth = 2 d Cos(th)

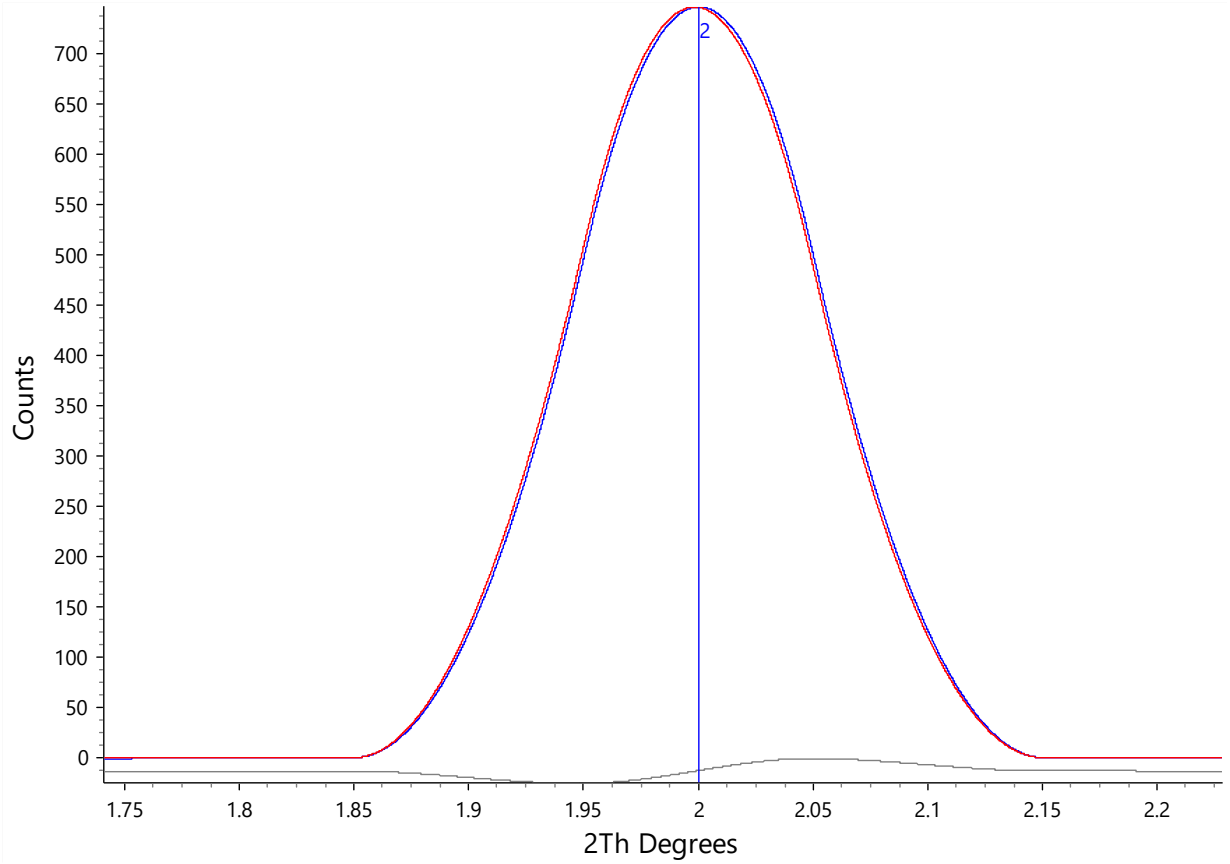
or,

I(th) = I(lam) 2 d Cos(th)

Rearranging we get:

I(th) = lam I(lam) Cot(th)

The point by point intensity of the emission profile therefore changes as function of Cot(th). DISP.INP show difference between correcting for and not correcting for dispersion as follows:



The peak shape of the above is as follows:

```
hat 0.1 num_hats 3 ' specimen/instrument
lam ymin_on_ymax 0.0000001 la 1 lo !lam_0 1.540596 lh 0.5
    modify_peak_eqn =
        Get(current_peak)
        If (And(Get(current_peak_x)>(lam_0-1.5),Get(current_peak_x)<(lam_0+ 1.5)),
            1 / Tan(ArcSin(Get(current_peak_x) / (2 D_spacing))),
            0
        );
     modify_peak_apply_before_convolutions
```

3. ..REFERENCES

Farrow, C.L; Juhas, P.; Liu, J.W.; Bryndin, D.; Božin, E.S.; Bloch, J.; Proffen, Th. Billinge, S.J.L. (2007). *J. Phys.*: *Condens. Matter* 19 (2007) 335219 (7pp)

Madelung, Erwin. *"Das elektrische Feld in Systemen von regelmäßig angeordneten Punktladungen."* Physikalische Zeitschrift, 19, 524–532 (1918).

Verlet, Loup (1967). "Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard–Jones Molecules". *Physical Review*. 159 (1): 98–103. Bibcode:1967PhRv..159...98V. doi:10.1103/PhysRev.159.98