TOPAS Version 5 – What's New

1 Contents

2 New kernel functionality

2.1 Convolution

2.1.1..... ft_conv

The keyword *ft_conv* describes a Fourier Transform (FT) 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 and is optional
   ft_x_axis_range = 40 lfwhm;
   ` Get(ft_0)
   ` FT_Break
```

The convolution theorem is used here multiplying the FT of a Gaussian by the FT of a Lorentzian. Were the Fourier transforms separately defined then 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 x-axis 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 and spherical_harmonics_hkl

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 *spherical_harmonics_hkl* used with and without *ft_conv*.

```
prm csl 50 min 3 max = Min(Val 2 + .1, 10000);
prm csg 50 min 3 max = Min(Val 2 + .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.

2.1.2..... 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 using *ft_conv.* The keyword *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 directly and with a 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 in particular shows how to convolute a function with discontinuities in x-axis space; ie.

```
user_defined_convolution = Exp(-20 X^2); min = -.2; max = .5;
```

The FT for functions with such discontinuities often cannot be described analytically and hence the usefulness of *user_defined_convolution.*

2.1.3.....FFT versus direct convolution

Typically a FFT convolution is quoted as comprising $O(N \log_2 N)$ operations (Cooley–Tukey algorithm for example) versus a direct convolution that comprises $O(N^2)$ operations, see https://ccrma.stanford.edu/~jos/ReviewFourier/FFT_Convolution_vs_Direct.html for example, and that direct convolution 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 ~< 256 to 512 as the comparison of the $O(N \log_2 N)$ versus $O(N^2)$ is invalid. 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 and in direct convolution this makes a total of 15 multiplies (Nr*Np) and not N^2 where $N/2 \leq (Nr+Np) \leq N$. To perform such a convolution with an FFT the number of operations is approximately $4*16*\log_2 16=256$ multiplies where 16 is the closest power of 2 to Nr+Np. 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 in particular many of the response functions have a small Nr; these include axial divergence, equatorial divergence, receiving slit width, capillary convolution, LPSD convolution and often sample penetration.

2.1.4.....Convolutions in general

TOPAS approximates the number of operations required for direct and FFT convolution and chooses the one with the smaller number of operations. In addition all direct convolutions are performed with peaks treated as straight line segments. Response functions are either straight line segments, analytical or both. The extra cost of the piece wise integration is small, approximately 3 (Nr+Np) operations, and the benefit is a high degree of accuracy.

Apart from *lor_fwhm* and *gauss_fwhm*, all of the convolutions described below have discontinuities in 2Th space; their associated Fourier transform therefore is difficult to describe. In cases where a FT is less demanding then a FFT is used after first calculating the aberration in 2Th space.

Response functions that are treated as straight line segments are:

```
user_defined_convolution
capillary_diameter_mm
lpsd_th2_angular_range_degrees
```

Response functions that are analytically convoluted with the straight line segments of the peak 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 treated analytically with the emission profile to form the base profile. Convolutions are calculated with a step size given by:

```
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 Nr:

```
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 for the little extra cost in calculating the extra response function points and with the benefit of not increasing Np*Nr. 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 most convolutions scale by (Nr * Np) except for convolutions that scale by N; these are always performed directly and they 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_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 the convolution and the aberration that uses it:

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 with or without a finite thickness
hat	Receiving slit width, sample tilt
stacked_hats_conv	Tube tails

2.2 WPPM

Examples referred to in this section reside in the test_examples\wppm directory.

2.2.1 ..... 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.

2.2.2 ..... 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?

2.2.3 ..... 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_correct_Is
```

The result is then interpolated back to 2Th space. Interpolations are scaled such that $I(s)ds = I(\theta)d\theta$ when *WPPM_correct_Is* is defined; the affects 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 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 and spherical_harmonics_hkl

Example s-sphere-1.inp uses *WPPM_ft_conv* to fit to a synthesized WPPM generated peak with identical results.

The following macros (written by Matteo Lenoi), as defined in TOPAS.INC, describes a log normal distribution:

```
WPPM_Cube_Ln_Normal
WPPM_Sphere_Ln_Normal
WPPM_Octahedron_Ln_Normal
```

Where for example WPPM_Octahedron_Ln_Normal is as follows:

```
macro WPPM_Octahedron_Ln_Normal(muc, muv, sigc, sigv)
```

```
{
   #m_argu muc
   #m_argu sigc
   If_Prm_Eqn_Rpt(muc, muv, min .1 max = Min(2 Val + .3, 100);)
   If_Prm_Eqn_Rpt(sigc, sigv, min .01 max = Min(2 Val + .01, 3);)
   WPPM_ft_conv =
      {
         def cga = Constant(Cos(Get(ga) Pi/180));
         def sga = Constant(Sin(Get(ga) Pi/180));
         def cal = Constant(Cos(Get(al) Pi/180));
         def cbe = Constant(Cos(Get(be) Pi/180));
         def aa = Constant(Get(a));
         def bb = Constant(Get(b));
         def cc = Constant(Get(c));
         def cv = Constant(Get(cell_volume));

         def wA = H / aa;
         def wB = (-H cga / aa + K / bb) / sga;
         def wC = ( (H bb cc (cal cga - cbe) + K aa cc (cbe cga - cal)) /
          sga + L aa bb sga) / cv;

         def A = D_spacing Max(wA, wB, wC);
         def B = D_spacing Max(Min(wA, wB), Min(wA, wC), Min(wB, wC));
         def C = D_spacing Min(wA, wB, wC);

         def H0 = 1;
         def H1 = If (A>=B+C,-3 A/Sqrt(2),- 3 (A + B + C)/Sqrt(8));
         def H2 = If (A>=B+C, 3 (A A - B B - C C)/2,-3(A A+(B-C)^2 - 2 A
              (B+C))/4);
         def H3 = If (A>=B+C, (-A^3 + 3 A (B B+C C) + 2(B^3 + C^3))/(2
              Sqrt(2)),(A^3 + B^3 + C^3-3 A B C)/Sqrt(2));
         def Kc = (A + B + C)/Sqrt(2);
         def u = CeV(muc, muv);
         def sig = CeV(sigc, sigv);

         fn M(n) = Exp(Min(n u + 0.5 n^2 sig, 600));
         fn wppm_Ln(kc) = Get(WPPM_Ln_k) + Ln(kc Get(WPPM_dL));
         fn q(Hn, n) {
            return
               Hn
               Erfc_Approx( ( wppm_Ln(Kc) - u - (3-n) sig^2) / (sig
                 Sqrt(2)))
               WPPM_L^n
               M(3-n);
            }
         return q(H0, 0) + q(H1, 1) + q(H2, 2) + q(H3, 3);
      }
      WPPM_break_on_small = 1e-7;
      WPPM_L_max 1000
      WPPM_th2_range = 30;
}
```

Example cube-ln-normal-1.inp can be used to test these macros. Lattice parameters appearing within the macros are made constant using Constant; thus these convolutions are made independent of lattice parameter changes and hence a separate convolution is 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.

2.3 Loading of INP files

2.3.1..... if {} else if {} else {}

An 'if' construct operational on the loading of INP files, see test_examples\zro2.inp. Loading operates on the pre-processed INP file; syntax is as follows:

```
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 the following:

```
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("\nThis wont be 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
  }
}
```

2.3.2..... existing_prm

existing_prm allows for the modification of an existing *prm*/*local* parameter, 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 the result:

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

The operators of +=, -=, *-, /= and ^= are allowed.

2.4 CIF

The following macros and Get's can be used to output data in CIF format; Red corresponds to new macros:

```
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 following 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 the following masked with excluded regions:

```
1 / Max(1, Yobs),    if SigmaYobs does not exist
1 / SigmaYobs^2,     if SigmaYobs does exist
```

If *weighting* is a function of YCalc etc... then it returns the last weighting calculated depending on *recal_weighting_on_iter*.

2.5 Rigid bodies

2.5.1..... Rigid body parameter errors propagated to fractional coordinates

Errors for fractional coordinates for sites defined as part of a rigid body are now propagated to the site fractional coordinates. The example rigid-errors\Aniline_I_100K_x.inp (by Simon Parsons) demonstrates equivalent refinements for the case of 1) using a rigid body and for the case 2) hand coding the fractional coordinates in terms of rigid body parameters but not in fact using a rigid body. Errors and convergence behaviour in both cases are identical. In particular case (2), which has many computer algebra equations, takes approximately the same time per iteration as case (1); this demonstrates that the computer algebra does not noticeably affect computational speed even in cases where its use is plentiful.

2.5.2..... 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 snap shot of the rigid body operations pertaining to the error. The following is an example of the information displayed:

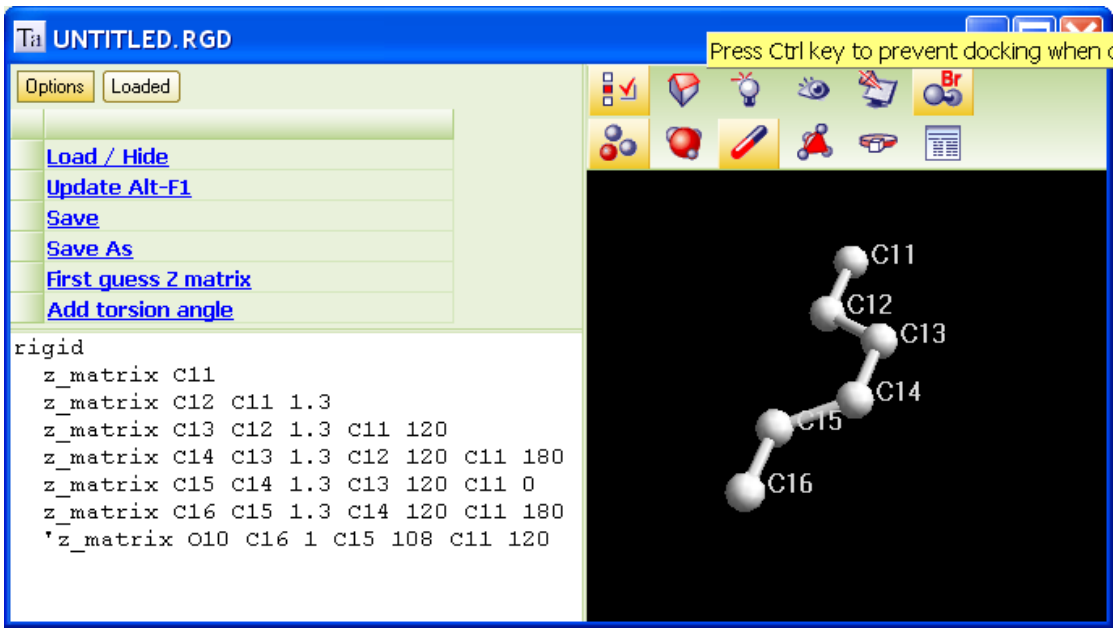
```
DB_x_CB Zero dot product - Z-matrix possible collinear points at atoms
```

```
            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
```

To investigate why the error is occurring the rigid body fragment can be copied to a Rigid-body editor window; ie.



The O10 line is commented out as it's the line in error. Looking at the O10 line (using the OpenGL window) it can be seen 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, ie.

```
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 DBC and hence the dihedral angle cannot be formed. Thus for z-matrices both A,B,C and B,C,D must not be collinear. Dummy atoms can solve this problem. The program tests for a zero dot product numerically with a tolerance of 1.0e-15.

2.6 Functions – fn, def, return, noinline

Functions can be defined using the 'fn' keyword; 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 with the ‘prm’ keyword; this simplifies the writing of ‘prm’ equations and additionally memory usage can be greatly reduced when the ‘noinline’ keyword 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 will recognize the ‘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(); : 6
```

‘def’ and ‘prm’ objects have scope and their scope determine 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 in both the function definition and its use.
- 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* as a stack mechanism is used for the *fn* arguments as well as the global simplification routines cannot simplify what's inside a *noinline*d 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 in 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 then not inlining may increase performance as the signalling 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);
```

2.6.1.....Subject independent single crystal refinement

The \functions\alvo4-fn.inp example performs a single crystal refinement using the computer algebra aspects of the program. No x-ray or subject dependent keywords have been used; instead only six keywords are utilized:

fn, noinline, def, return, prm, restraint

The speed of alvo4-fn.inp is 7.4 times slower than the comparable subject dependent keyword 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 behaviour of the parameters are similar. The placement of noinline is important. Also used is the *out_refinement_stats* keyword 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…

2.6.2.....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 reworked 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, outputs 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
```

2.7 The Minimization Routines

The Newton-Raphson non-linear least squares method is used by default with the Marquardt method (1963) included for stability. A Bound Constrained Conjugate Gradient (BCCG) method (Coelho, 2005) incorporating *min*/*max* limits is used for solving the normal equations. The objective function χ^2 is written as:

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

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

$$\chi_P^2 = K\, K_1 K_P \sum_{p=1}^{N_P} P_p$$

$$\chi_R^2 = K\ K_1\ K_R \sum_{r=1}^{N_R} R_r^2$$

$$K = 1 \Big/ \sum_{m=1}^{M} w_m\ Y_{o,m}^2 \tag{2-3}$$

$Y_{o,m}$ 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 the keyword *penalty*, and N_p the number of penalty

functions. R_r are restraints, defined using the keyword *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 normal 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 normal equations are:

$$\mathbf{A}\,\Delta\mathbf{p} = \mathbf{Y} \tag{2-4}$$

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

$\mathbf{Y} = \mathbf{Y}_0 + \mathbf{Y}_P + \mathbf{Y}_R$

$$A_{0,ij} = \sum_{m=1}^{M} w_m \frac{\partial Y_{c,m}}{\partial p_i}\frac{\partial Y_{c,m}}{\partial p_j} \tag{2-5}$$

$$A_{P,ij} = K_P \frac{1}{2}\sum_{p=1}^{N_P}\frac{\partial^2 P_P}{\partial p_i \partial p_j}$$

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

$$Y_{0,i} = \sum_{m=1}^{M} w_m (Y_{o,m} - Y_{c,m})\frac{\partial Y_{c,m}}{\partial p_i}$$

$$Y_{P,i} = -\frac{K_P}{2}\sum_{p=1}^{N_P}\frac{\partial P_P}{\partial p_i}$$

$$Y_{R,i} = -K_R \sum_{r=1}^{N_R} R_r \frac{\partial R_r}{\partial p_i}$$

The Taylor coefficients $\Delta\mathbf{p}$ correspond to changes in the parameters **p**. Eq. (2-4) 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_0^2 and χ_P^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. (2-6).

$$K_P = \sum_{k}^{N_k} \mathrm{If}\,(Y_{P,kk} < 10^{-14} A_{0,kk}, 0, 1.05\, A_{0,kk} / (A_{P,kk} + A_{0,kk}\ \mathrm{Min}(Y_{P,kk}/Y_{0,kk}, 0.05))) \tag{2-6}$$

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

$$K_R = \sum_{k}^{N_k} \mathrm{If}\,(Y_{R,kk} < 10^{-14} A_{0,kk}, 0, 1.05\, A_{0,kk} / (A_{R,kk} + A_{0,kk}\ \mathrm{Min}(Y_{R,kk}/Y_{0,kk}, 0.05))) \tag{2-7}$$

K_R and K_P can be modified using *pen_weight* and the macro Pen_Wt. Pen_Wt calls the macro Write_Pen_Wt which then has to be defined by the user. A definition that mimics the default is as follows:

```
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 $Y_{P,i}$.

To formulate ShelX type restraints the following could be used:

```
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.

2.7.1.....Improvements to Conjugate Gradient Solution method

The bound constrained conjugate gradient method (Coelho, 2005) used for solving the normal equations greatly assists in convergence of the non-linear least squares process. Previously min/max limits were calculated prior to the solution of the normal equations and then held constant during the solution process. Version 5 in addition dynamically recalculates min/max limits during the solution process for min/max limits that are a function of independent parameters. For example, to constrain site occupancies on three sites to full occupancy with three atomic species each with occupancy of 1 the following could be defined:

```
site Ni  x .11 y .22 z .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 .21 y .32 z .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 .31 y .42 z .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
```

Version 4 allowed for such constraints but had difficulty in finding a minima without violating the limits. Version 5 has no such difficulty as seen in test_examples\occ-constrain.inp.

2.7.2.....Restraints and Penalties

A particular restraint can be reformulated into a penalty by squaring the restraint, for example:

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

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 faster than equivalent penalties due to the use of the off-diagonal terms (compare ROSENBROCK-10.INP with ROSENBROCK-10-RESTRAINT.INP). 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 $\mathbf{A}_R$ 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 diagonal elements of $\mathbf{A}_0$, $\mathbf{A}_P$, and $\mathbf{A}_R$ are not calculated; instead they are approximated by the BFGS method.

When *approximate_A* is used and both penalties and restraints are defined then this effectively means that the restraints are treated as penalties. The following for example:

```
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);
```

will have similar but not identical convergence to the following:

```
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;
```

In Case 1 the diagonal elements of the A matrices are:

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

$A_{R,r1r1} = (\partial R1/\partial r1)^2 + (\partial R2/\partial r1)^2$

In Case 2 they are:

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

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

The difference in behavior between penalties and restraints can be seen by comparing ROSENBROCK-10.INP to ROSENBROCK-10-RESTRAINT.INP. In 500,000 iterations we have:

```
ROSENBROCK-10.INP:  71 iterations on average to convergence
ROSENBROCK-10-RESTRAINT.INP: 47 iterations on average to convergence
```

The restraints converge faster as the $A_{R,ij}$ elements are calculated. Approximating $A_{R,ij}$ by defining *approximate_A* in ROSENBROCK-10-RESTRAINT.INP gives the fastest convergence time wise:

```
ROSENBROCK-10-RESTRAINT.INP: 71 iterations on average to convergence
```

Many penalties however cannot be formulated as a restraint, RASTRIGIN.INP for example, and in these cases penalties are mandatory.

2.7.3.....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 Chi^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 Chi^2

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

2.7.4.....Error calculation

Errors are calculated for all independent and non-independent parameters that are single valued when any of the following is defined:

`do_errors`: Errors calculated without the inclusion of penalties and restraints in the A matrix.

`do_errors_include_penalties`: Errors calculated with the inclusion of penalties in the A matrix.

`do_errors_include_restraints`: Errors calculated with the inclusion of restraints in the A matrix.

2.7.5.....Simulated annealing adaptive step size

The adaptive step size used in simulated annealing has been improved. In many case the complex temperature regime found in the macro Auto_T can be replaced with a single temperature. The example CIME-Z-AUTO.INP demonstrates the improvements by using a very incorrect starting temperature of 0.1; the program quickly modifies the temperature to a more appropriate vale. Output lines such as:

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

indicate that a particular 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.

2.7.6.....Refining on an arbitrary Chi^2

The *chi2* keyword allows for minimization of a user defined χ^2. It can be a function of the reserved parameter names X, Yobs, Ycalc and SigmaYobs. In addition the keyword *xdd_sum* is a parameter that 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 minima in *chi2* is bracketed and inverse parabolic interpolation 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 *test_example\chi2-ceo2.inp* example 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.

2.7.7.....Informing of unrefined parameters

Parameters that do not take part in a refinement are now 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 refinement will result in the following 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,

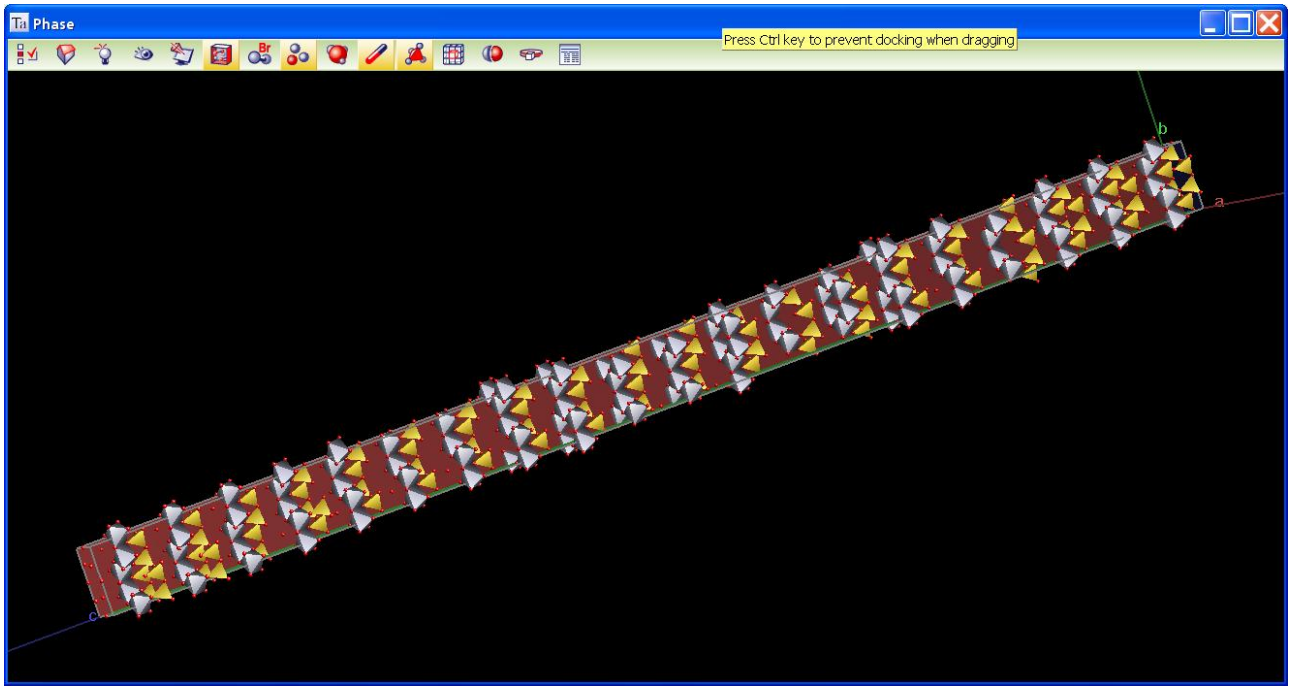
```
prm a 1 val_on_continue = b = 2; ' this sets the parameter b to 2.
```

2.8 Stacking faults

```
[site $name]...
   [layer $layer_name]
[stack $layer_name]...
   [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_name*. *stack* applies a stacking vector (*sx*, *sy*, *sz*) to the named layer. Structures factors are generated in the usual manner; a shift corresponding to the stacking vector is then applied. *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; ie.



Importantly penalties operate on dummy sites which allow restraints such as Distance_Restrain. For example,

```
space_group P1
site O1... layer A
site O2... layer A
stack A
   sx...
   generate_these O1
      generate_name_append _1
append_fractional
   in_str_format
```

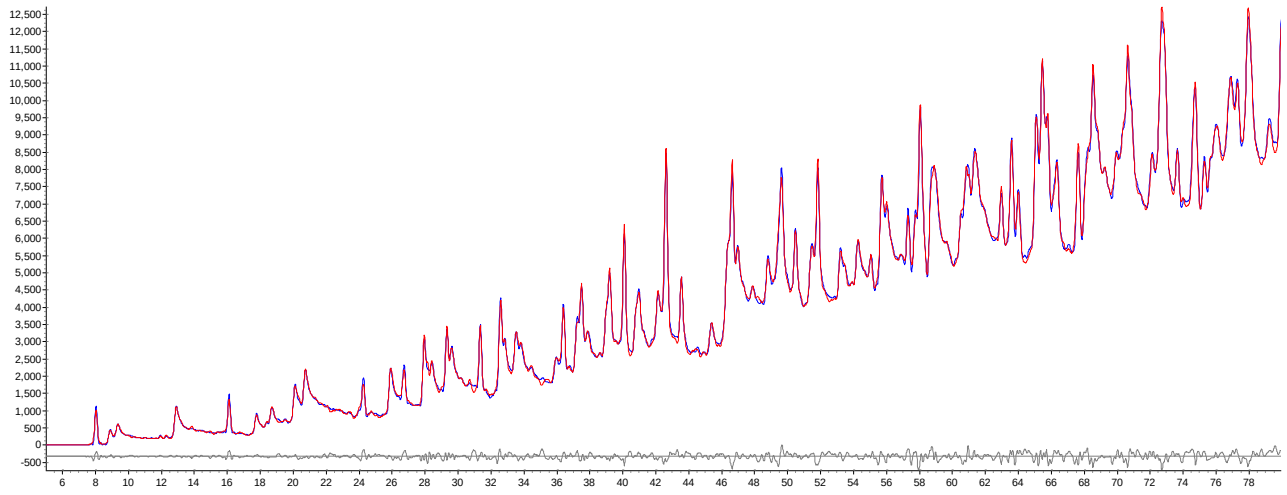
will output for append_fractional the following:

```
site O1 ...
site O2...
site O1_1 ... occ O 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.

2.8.1.....Fitting to a Debye-formulae generated 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 new phase dependent keyword called [*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. A particular I(2Th) intensity has its intensity distributed to the two end peaks as follows:

$$I(2Th_1) = I(2T)\ f$$
$$I(2Th_2) = I(2T)\ (1 - f)$$

where,

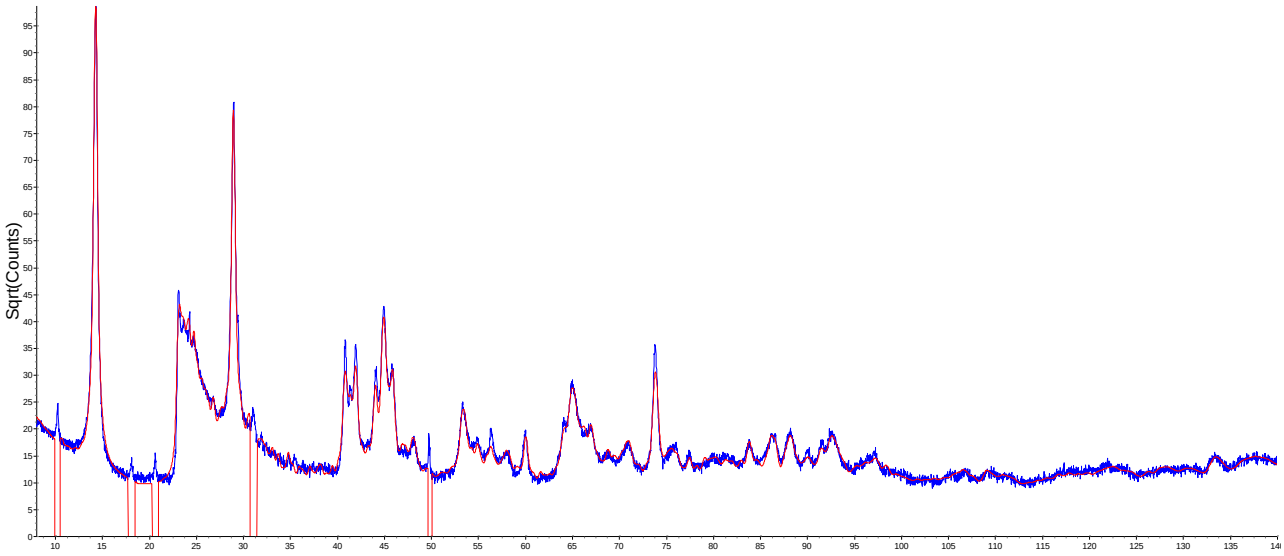
$$f = (\cos(\pi(2Th - 2\ Th_1) / (2\ Th_2 - 2\ Th_1)) + 1) / 2$$

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 debye-new.inp) is time consuming; turning the graphical hkl ticks option Off is worthwhile.

2.8.2.....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.

2.9 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
```

2.10 Quantitative Analysis

The following are items associated with quantitative analysis with new items in red:

```
xdd..
   mixture_MAC
   mixture_density_g_on_cm3
   weight_percent_amorphous
   elemental_composition
   element_weight_percent...
   element_weight_percent_known...
   prm = Get(sum_smvs)...
   Mixture_LAC_1_on_cm(0) ' macro
   str...
      weight_percent
      cell_mass
      cell_volume
      phase_MAC
      spiked_phase_measured_weight_percent
      corrected_weight_percent
      prm = Get(sum_smvs)...
      prm = Get(smv)...
      prm = Get(sum_smvs_minus_this)...
      prm = Get_Element_Weight(atom)...
      Phase_LAC_1_on_cm(0) ' macro
      Phase_Density_g_on_cm3(0) ' macro
```

Test_examples\quant\quant-1.inp uses many of these and additionally writes equivalent terms in the form of equations, for example:

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

   prm q = spiked_phase_measured_weight_percent /
            spiked_phase_measured_weight_percent_wt; : 0

   ' This is corrected_weight_percent
   prm = q Get(weight_percent); : 0

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

2.10.1... Elemental weight percent constraint

The QUANT implementation is almost entirely written internally using the TOPAS Symbolic system. Dependencies are automatically taken care of and unnecessary recalculations kept to a minimum. The overriding plus however is the flexibility it allows. If for example 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; ie.

```
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 follows:

```
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. The test_example\quant\quant-3.inp demonstrates this constraint with very good convergence. It comprises 4 phases with three of them 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));
    }
```

2.10.2... Elemental composition and Restraints

The *xdd* dependent keyword *element_composition* reports 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
        }
```

element_weight_percent $ELEMENT $NAME #: is an *xdd* dependent keyword that returns the weight percent of an element within the corresponding *str*'s of the *xdd*. Example usage:

Before Refinement:

```
penalties_weighting_K1 .1
xdd...
    element_weight_percent Zr+4 zr  0
    restraint = (zr - 65); :  0
```

After Refinement:

```
penalties_weighting_K1 .1
xdd...
    element_weight_percent Zr+4 zr  65.0275252`
    restraint = (zr - 65); :  0.0275251892`
```

In this example zr is the name given to the element Zr+4 and 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*.

For restraining a weight percent the following can be used:

```
xdd...
    penalties_weighting_K1 .2
    restraint = (Cubic_Zirconia_wt_percent - 36); : 0
    str...
        MVW(0,0, !Cubic_Zirconia_wt_percent 0)
```

Note, the Cubic_Zirconia_wt_percent name which is given to the weight percent; see test examples\quant.

2.10.3... Amorphous phase composition

If *spiked_phase_measured_weight_percent* is defined then *elemental_composition* will report on Rietveld values, Corrected values as well as values from the original un-spiked sample. If *element_weight_percent_known* keywords are defined then e*lemental_composition* will additionally report on the elemental contents of the amorphous phase, for example, from test_examples\quant\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 first second and third columns sum to 100%. The second column corresponds to corrected values including the spiked phase. The third and fourth columns relate to elemental weight percents of the original phase. The last row of the second column (in purple) corresponds to Get(*weight_percent_amorphous*), the last row of the fourth column (in red) is the amount that is undefined; it comprises the Green number minus the elements of the third column excluding the last row. Note the zeros for Al (in blue); this is due to the spiked phase (dummy test data) being the only phase containing Al.

2.10.4... 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 as follows:

```
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)
```

*** Note: *dummy_str*'s void of MVW and sites takes no part in Quantitative analysis.

The lattice parameters and the chemistry should correspond to a real structure in order for Mixture_LAC_1_on_cm and phase_LAC to 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 other than 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 *dummy_str* has been assigned the amorphous content. 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
}
```

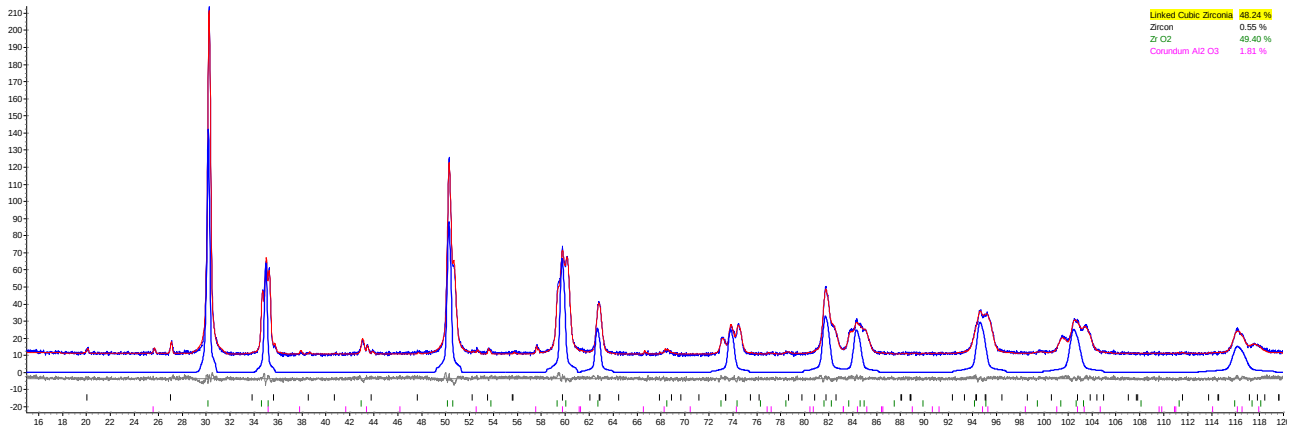
The negative element weight percent for O for the amorphous content reflects the fact that the measured XRF value for O is lower than the refinement's value (note, this example is simply for testing and the XRF values used are fictitious).

2.10.5... 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 unknown but the chemistry is known and a calibration constant has been determined relating the hkl_Is intensities to the scale parameter of the *hkl_Is* phase. In such a case the *dummy_str* can be written as (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  0.962
   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 occur for example 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 it scale parameter to the *hkl_Is* scale parameter. Quant-7.inp is a similar process 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.

2.10.6...Summary of Quant examples

- Quant-1.inp: a general example showing 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 an *hkl_Is* phase instead of a *str* phase; uses Known_Weight_Percent on the *hkl_Is* phase.
- Quant-5.inp: uses a *dummy_str* to describe an amorphous phase
- Quant-6.inp: uses a *hkl_Is* phase to describe a phase; links a *dummy_str* to the *hkl_Is* phase to get QUANT info.
- Quant-7.inp: uses a *fit_obj* that is a function of a *user_y* object to describe a phase; links a *dummy_str* the *fit_obj* to get QUANT info.

2.10.7...External standard method

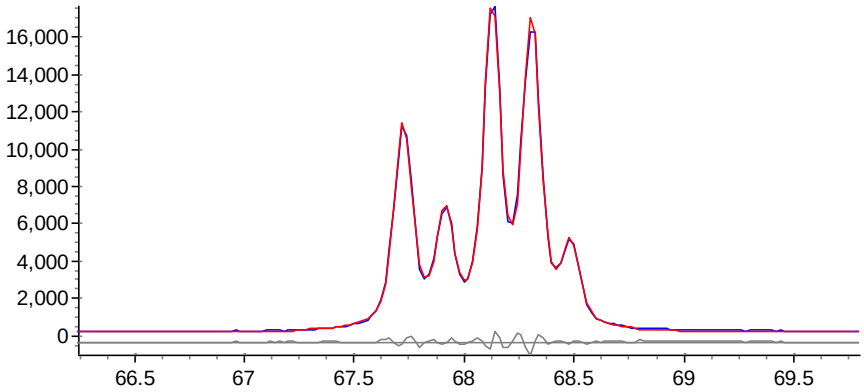
The method of O'Connor and Raven (1988) has been implemented in both GUI and Launch modes through the use of the macros:

```
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_;
    }
}
```

See test_examples\k-factor.

2.11 Learnt Shapes for Background or Otherwise

The new keywords *user_y* and *fo_transform_X* provides a means to use learnt shapes as a background function. The test example USER_Y.INP produces the following fit to Quartz using a learnt Pseudo-Voigt.



Example usage:

```
user_y NAME { #include SOME_FILE }
user_y NAME SOME_FILE
fit_obj = NAME;
   fo_transform_X = (X - x) / s;
```

The user defined NAME corresponds to a parameter name given to the *user_y*; it can be used in all equations that can be a function of X, for example:

```
fit_obj = Exp(NAME^2);
```

fo_transform_X is a dependent of *fit_obj* and it transforms the X used within the *fit_obj*. For example, NAME could have an x-axis that does not match the x-axis of the Yobs pattern; *fo_transform_X* provides a means to transform the Yobs x-axis to the *user_y* 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 :

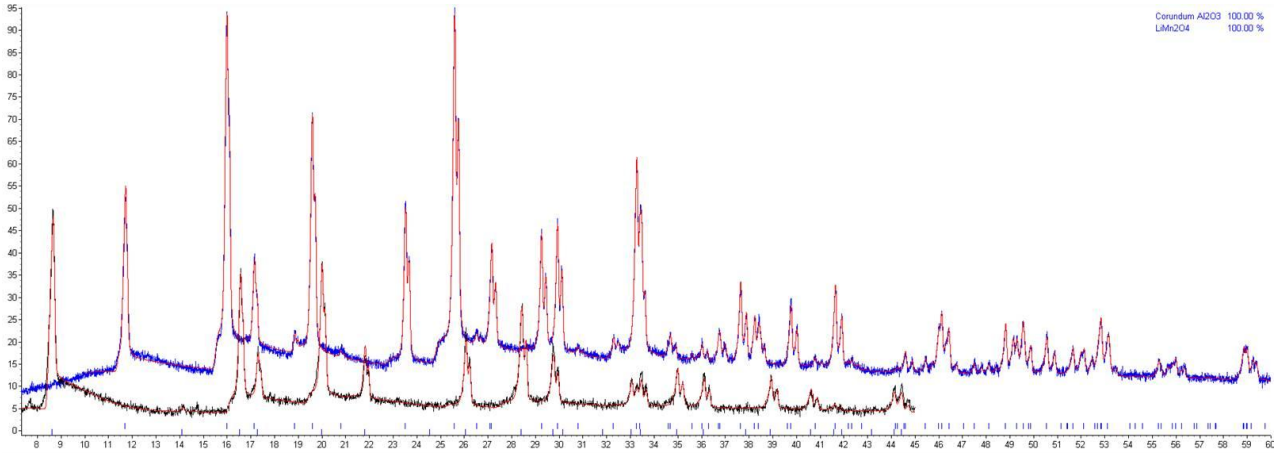
```
user_y NAME
   {
      _x1_dx -1 1 /* the start x and step */
      0 1 0       /* the shape data       */
   }
```

More than one *user_y* can be defined and they 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 other refinement.

2.12 Emission Profile with Absorption Edges

```
lam…
   [modify_peak]
      [modify_peak_apply_before_convolutions]
      [modify_peak_eqn !E] : Can be a function of Get(current_peak) and
                             Get(current_peak_x)
         [current_peak_min_x !E]
         [current_peak_max_x !E]
```

The *modify_peak* keyword can be used to modify peak profiles either before convolutions or after; here's a plot from AL2O3-Spinnel-PAM.INP (see directory test_examples\absorption-edge) that has an identical absorption edge modelled for both Al2O3 and Spinnel samples:



modify_peak 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 worked on by the program and the current calculated peak intensity at that x-axis position respectively.

2.13 scale_phase_X keyword

The *scale_phase_X* keyword scales Ycalc point by point. It can be used to define say alternate Lorentz Polarization factors. Some main points:

- Can be a function of X
- Multiple definitions are allowed and each is 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_Is* phase is multiplied by the first and third *scale_phase_X*.

2.14 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 example 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:

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

where the superscript * denotes conjugate gradient and:

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

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

Qhat = Q / |**Q**|

where

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 ^{T} denotes matrix transpose
$(\mathbf{L}^{-1})^{T}$ = 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:

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

where the summation is over the equivalent positions j and

$U_j = \mathbf{h}.\mathbf{R}_j \mathbf{x} + \mathbf{h}.\mathbf{t}_j$
x = { x, y, z } = site fractional coordinates
m = { mlx, mly, mlz } = magnetic moment
$\mathbf{R}_j$ = rotation part of space group operator
$\mathbf{t}_j$ = translational part of space group operator
$d_j = s_j$ determinant($\mathbf{R}_j$) = s_j det($\mathbf{R}_j$)
$\mathbf{B}_j = s_j$ det($\mathbf{R}_j$) $\mathbf{R}_j$ = magnetic transformation matrix

The file MAGDATA.DAT (a GSAS file - permission for its use granted from Robert Von Dreele, the 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.

mag_only: When defined the x-ray component to intensity for the site in question is ignored.

mag_only_for_mag_sites: When defined the x-ray component to intensity for all magnetic sites for the str in question is ignored.

2.14.1... Magnetic refinement warnings/exceptions

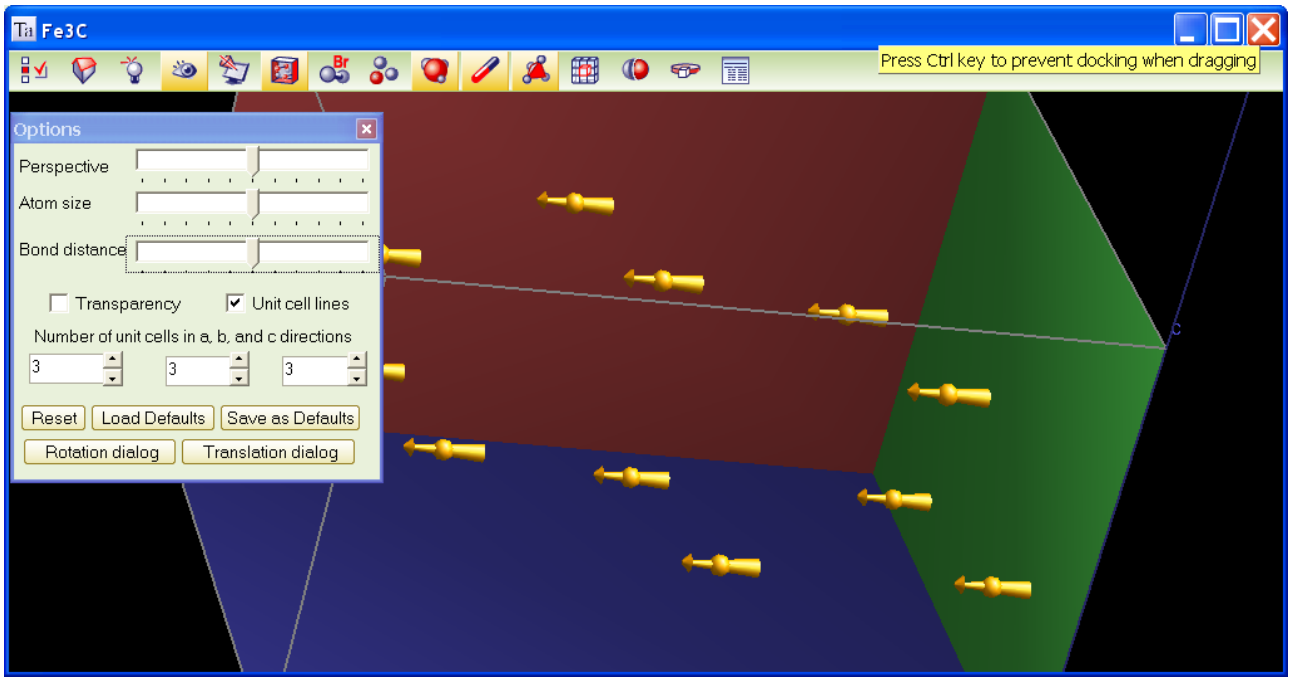
The following two messages:

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.

2.14.2…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:



2.14.3…'Decomposing' Fmag for speed

When using magnetic space groups other than 1.1 equivalent positions of the space group are written in terms of other equivalent positions.

Let

$C_j = \cos(U_j)$

$S_j = \sin(U_j)$

$\mathrm{Exp}(i\,U) = C_j + i\,S_j$ = Euler's formulae

For two equivalent positions of a special position we have

$U_1 = U_2 = U$

$\mathbf{Fmag}_1 + \mathbf{Fmag}_2 = s_1 \det(\mathbf{R}_1)\ \mathbf{R}_1\ \mathbf{m}\ \mathrm{Exp}(i\,U) + s_2 \det(\mathbf{R}_2)\ \mathbf{R}_2\ \mathbf{m}\ \mathrm{Exp}(i\,U)$

$= (s_1 \det(\mathbf{R}_1)\ \mathbf{R}_1 + s_2 \det(\mathbf{R}_2)\ \mathbf{R}_2)\ \mathbf{m}\ \mathrm{Exp}(i\,U)$

$= \mathbf{c}\ \mathbf{m}\ \mathrm{Exp}(i\,U)$

c is independent of **x**

Note, a particular special position could have many equivalent positions.

If $\mathbf{R}_1 = -\mathbf{R}_2$ and $\mathbf{t}_1 = -\mathbf{t}_2$ for two equivalent positions then

$U_1 = -U_2 = U$

$\mathbf{Fmag}_1+\mathbf{Fmag}_2 = s_1 \det(\mathbf{R})\ \mathbf{R}\ \mathbf{m}\ \mathrm{Exp}(i\ U) + s_2 \det(-\mathbf{R})\ (-\mathbf{R})\ \mathbf{m}\ \mathrm{Exp}(-i\ U)$

Now,

$\det(\mathbf{R})\ \mathbf{R} = \det(-\mathbf{R})\ (-\mathbf{R})$

or,

$\mathbf{Fmag}_1+\mathbf{Fmag}_2 = \det(\mathbf{R})\ \mathbf{R}\ \mathbf{m}\ (s_1 \mathrm{Exp}(i\ U) + s_2 \mathrm{Exp}(-i\ U))$

For $s_1 = s_2$

$\mathbf{Fmag}_1+\mathbf{Fmag}_2 = s_1 \det(\mathbf{R})\ \mathbf{R}\ \mathbf{m}\ 2\ C$

For $s_1 = -s_2$

$\mathbf{Fmag}_1+\mathbf{Fmag}_2 = s_1 \det(\mathbf{R})\ \mathbf{R}\ \mathbf{m}\ (2\ i\ S)$

If $\mathbf{R}_1 = \mathbf{R}_2$ for two equivalent positions then

$\mathbf{Fmag}_1+\mathbf{Fmag}_2 = s_1 \det(\mathbf{R})\ \mathbf{R}\ \mathbf{m}\ \mathrm{Exp}(i\ \mathbf{h}.\ \mathbf{R}\ \mathbf{x})\ \mathrm{Exp}(i\ \mathbf{h}.\mathbf{t}_1) + s_2 \det(\mathbf{R})\ \mathbf{R}\ \mathbf{m}\ \mathrm{Exp}(i\ \mathbf{h}.\ \mathbf{R}\ \mathbf{x})\ \mathrm{Exp}(i\ \mathbf{h}.\mathbf{t}_2)$

$= \det(\mathbf{R})\ \mathbf{R}\ \mathbf{m}\ (s_1 \mathrm{Exp}(i\ \mathbf{h}.\mathbf{t}_1) + s_2 \mathrm{Exp}(i\ \mathbf{h}.\mathbf{t}_2))\ \mathrm{Exp}(i\ \mathbf{h}.\ \mathbf{R}\ \mathbf{x})$

$= \mathbf{c}\ \mathrm{Exp}(i\ \mathbf{h}.\ \mathbf{R}\ \mathbf{x})$

c is independent of **x** and is calculated only once.

Many **R**'s can be the same for a particular space group with only the **t**'s changing.

Calculating C and S

Exp(i (**h** . **R x** + **h**. **t**)) = Exp(i **h** . **R x)** Exp(i **h** . **t**)

Exp(i **h** . **t**) is constant for a particular **h** and is calculated only once.

Only unique Exp(i **h** . **R 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 is not calculated for each h.

Note a sin or cos function is equivalent to about 40 to 60 multiplies.

2.15 Refining on f0, f' and f''

```
[f0_f1_f11_atom]…
   [f0 E] [f1 E] [f11 E]
```

Example usage is as follows:

```
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* and *f11*, *scale* and *beq* parameters.

The *f0_f1_f11_atom* keyword can be used at the *str*, *xdd* and global levels. f' or f'' can be defined and refined independently. Defaults are used when either f' or f'' are not defined. The XRAY-POWDER.INP and TOF.INP in the directory test_examples\f0-f1-f11\ 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 the *f0* parameter corresponds to the neutron scattering length. Defaults for neutron scattering lengths are obtained from the file neutscat.cpp. Neutron scattering lengths can be refined as demonstrated in test_examples\f0-f1-f11\TOF.INP.

- Keyword *no_f11* instructs the program to ignore *f11*. This increases speed with little change in Ycalc.
- Keyword *report_on_str* reports on *f1* and *f11* or neutron scattering lengths used. No values are reported when the keyword *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
}
```

2.15.1... Invalid f1 and f11

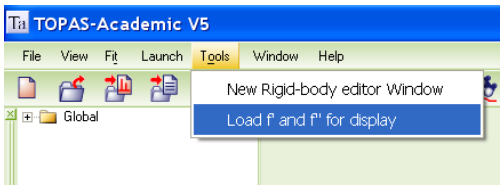
The following message is displayed when there are no valid entries for f' and f'' in the corresponding NFF file:

```
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* keywords respectively, see test_examples\f0-f1-f11 directory. Also of use is to view f' and f'' NFF files found in the SSF directory using the GUI Tool menu; ie.



2.16 Isotopes and Atom Names

The file mac\tab1.html is no longer used. Instead isotopes.txt is used for obtaining isotope weights. It's now possible to have the following when refining either neutron (ie. the keyword 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 isotopes.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. The numbers occurring at the start of the symbol is 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 (ie. ‘+2’ in this case) is dropped.

In the case of neutrons:

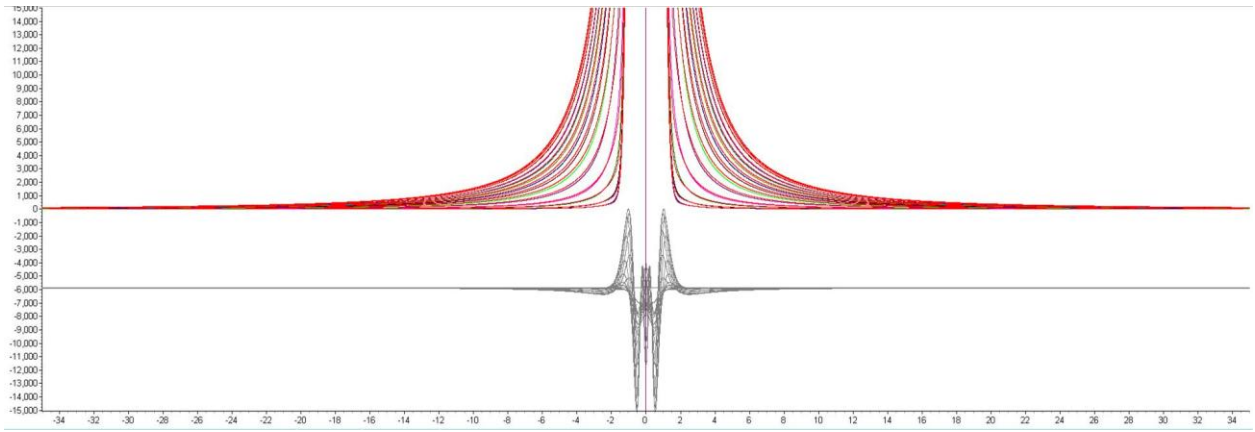
- scattering lengths used are from the neutscat.cpp file; the charge (ie. ‘+2’) is dropped when searching neutscat.cpp.

Internally the program converts ‘D’ and ‘T’ to ‘2H’ and ‘3H’ respectively.

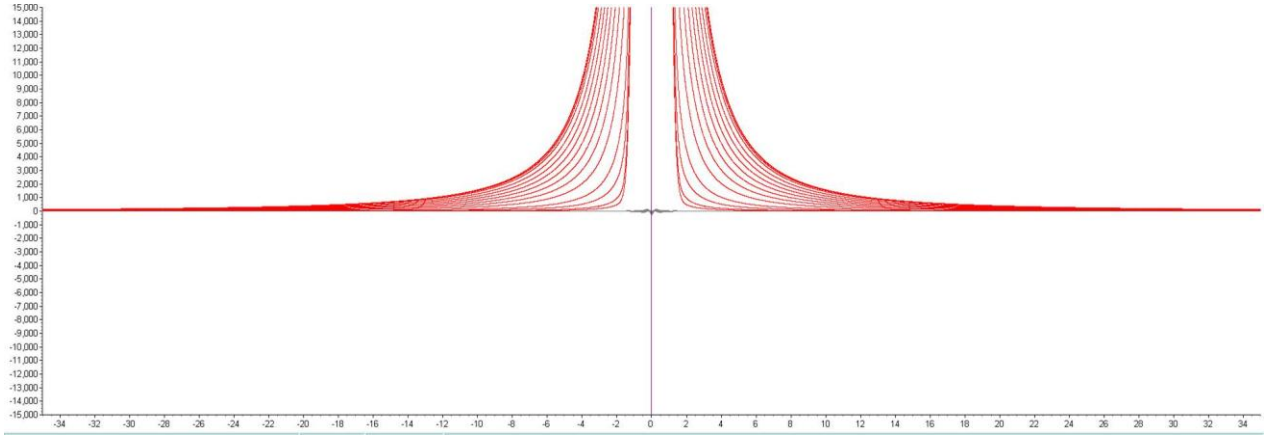
2.17 An Accurate Voigt

The *more_accurate_Voigt* keyword can be used to over ride the default Pseudo-Voigt approximation to the Voigt. The *more_accurate_Voigt* keyword decreases the error (Voigt_approx – Voigt_true) by a factor of around 100. Defining G as the full width at half maximum (FWHM) of a Gaussian and L for the FWHM of a Lorentzian the screen shots below are fits to a range of G convoluted with L (Voigts) with L varying from 0.01 to 0.09 and G+L=1.

Fitting to the Voigts using pseudo-Voigts:

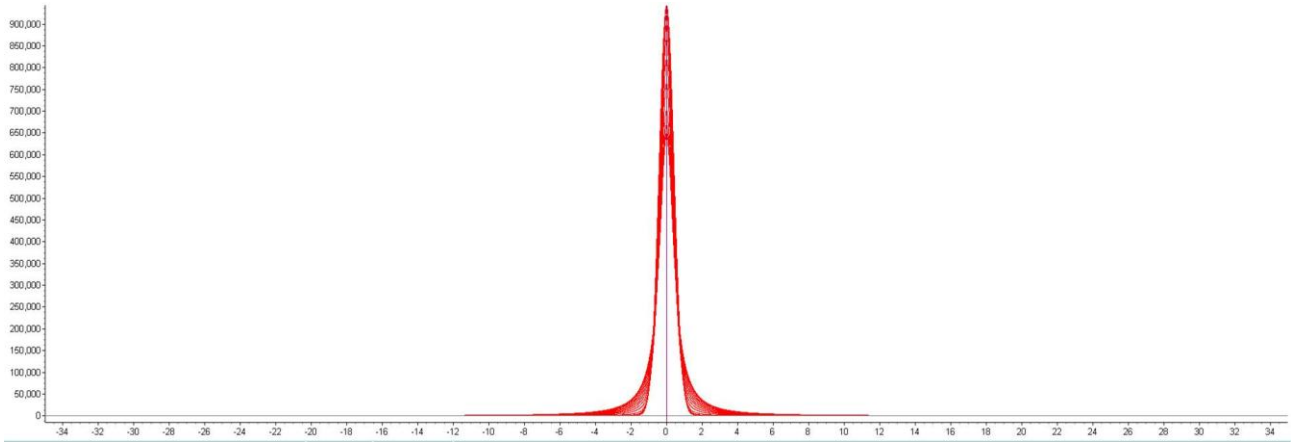


Fitting to the Voigts using an accurate calibration:



Note the very small difference-plots for the accurate calibration.

Rescaling the plot vertically to show the whole scan we have:



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.

The CREATE.INP file in the TEST_EXAMPLES\VOIGT-APPROX\ directory creates a true Voigt. It uses the keyword *numerical_lor_gauss_conv*. The amount of Lorentzian is entered as a number out of a 1000. A number of 500 say would yield a Voigt with a Laurentzian 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 actually 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 the c++ equivalent of 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.

2.18 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.

2.19 Atomic data files and associated sources

Table **2-1** lists the 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.

Table 2-1 Files and associated sources for atomic data.

File	Comment
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	f_0 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

2.20 Removing Phases during a refinement

The *remove_phase* keyword (used by the Remove_Phase macro) allows for phase removal during refinement. Typical use is as follows:

```
for strs {
   Remove_Phase(0.3, 1)
}
```

Here a phase is removed if its weight percent is below 0.3% and if the error in the weight percent is greater than 1%. 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 phase is removed during a Cycle.

2.21 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 when lor_fwhm and gauss_fwhm are defined within push_peak and add_pop_1st_2nd_peak keywords they are still calculated at the emission profile level.

3 New GUI functionality

3.1 Plotting phases above background

By default phases are now plotted on top of back ground where back ground comprises *fit_obj*'s+*bkg*. The *xdd* dependent keyword *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 back ground.

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*.

3.2 Plotting fit_objs

fit_obj's can be plotted using the following macros:

```
macro Plot_Fit_Obj(p, name)
   {
      dummy_str
         phase_name name
         scale = p;
   }

macro Plot_Fit_Obj(name)
   {
      dummy_str
         phase_name name
   }
```

See test_examples\voigt-approx\fit-obj.inp for example; ie.

```
xdd...
   fit_obj !f1 = ...
    Plot_Fit_Obj(f1, "Fit Obj")
```

Plotting is via a *dummy_str* and the scale parameter of the *dummy_str* is 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.

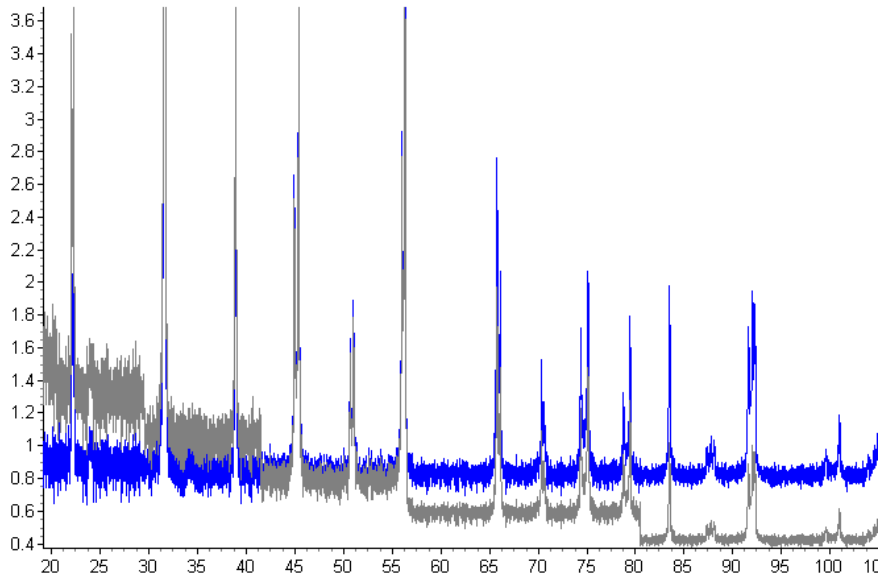
Sometimes the *fit_obj* has no name and no parameter that belongs to it; instead of naming the *fit_obj* or rearranging *prm* definitions the second Plot_Fit_Obj macro can be used:

```
xdd...
   fit_obj =
    Plot_Fit_Obj("plot previously defined fit_obj")
```

Here the *fit_obj* defined prior to Plot_Fit_Obj is plotted.

3.3 Display of Normalized SigmaYobs^2

Useful for checking SigmaYobs anomalies from VCT or XYE files; here's an example :



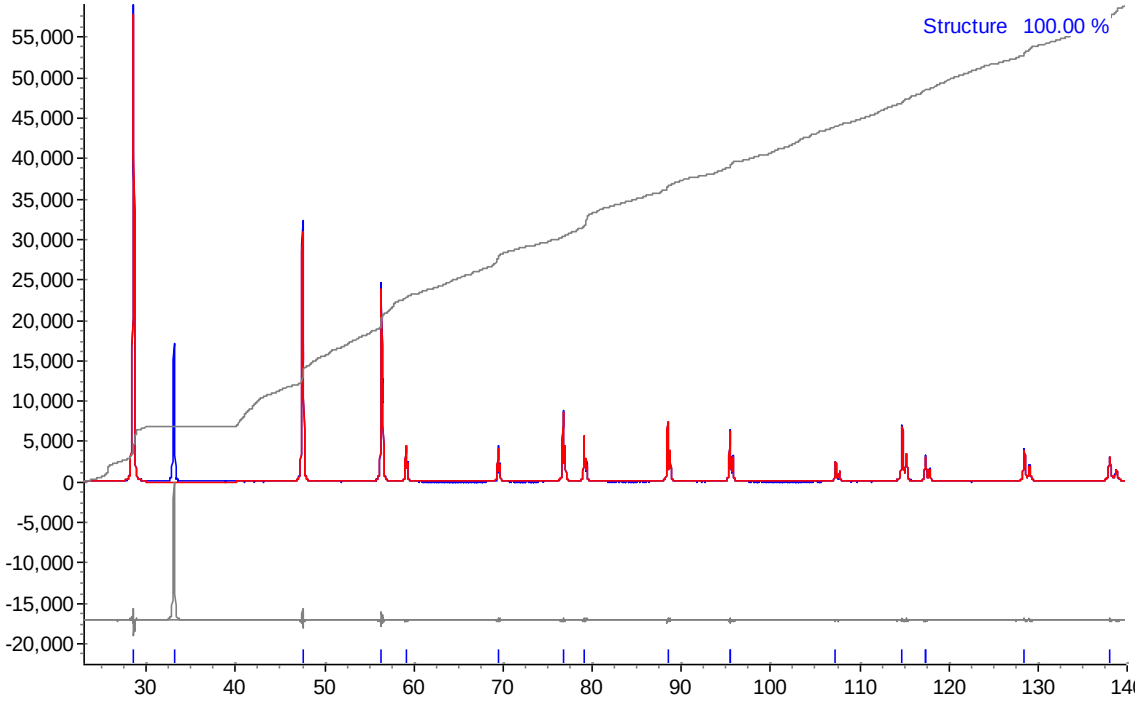
The normalization is as follows:

```
SigmaYobs^2 displayed = SigmaYobs^2 Sum[ Yobs ] / Sum[ SigmaYobs^2]
```

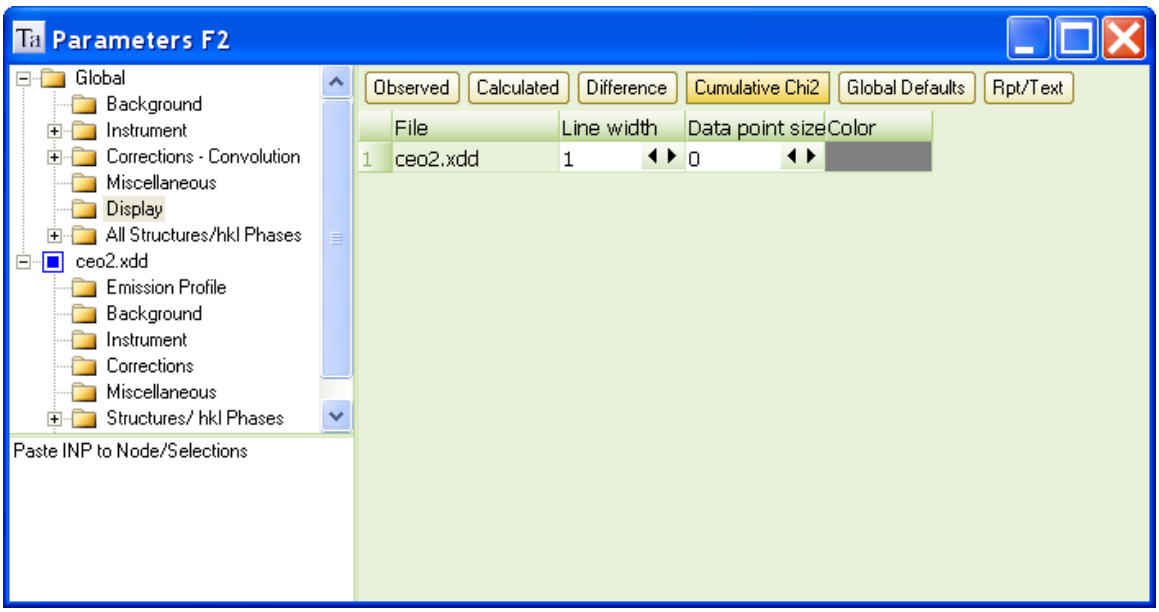
This puts the display of SigmaYobs^2 on a similar scale to Yobs. For normal x-ray data SigmaYobs=Sqrt(Yobs) and hence nothing is done as the displayed plot would simply be equal to Yobs. On some data sets, TOF for example, the magnitude of SigmaYobs can be small; thus when refining on multiple data sets from different sources the weighting schemes may need to be modified in order to give the desired weight to the data sets. The option for display is as follows::

3.4 Cumulative Chi2

A kernel operation that results in the following graphical display:

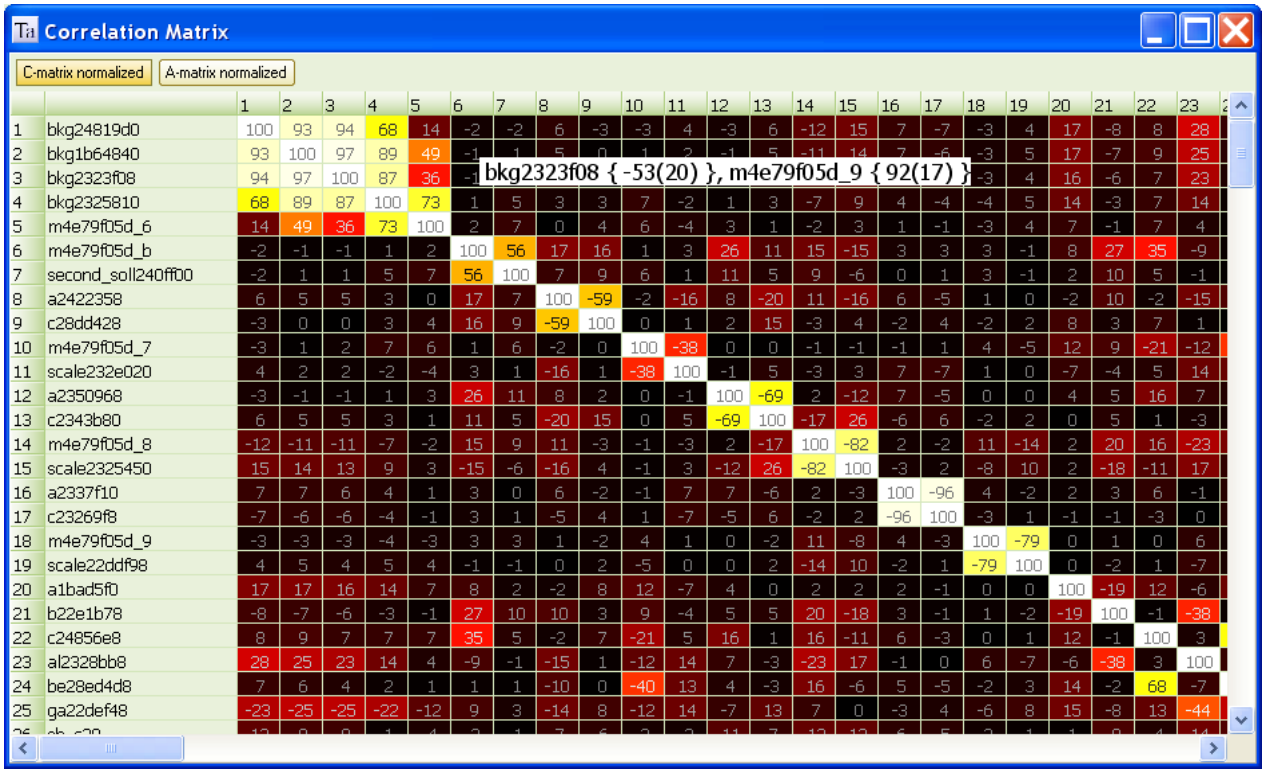


- Uses the weighting from the kernel which can be user defined or otherwise.
- SigmaYobs is used in the weighting if it exists.
- Prior to graphical display it is scaled to have the same maximum intensity as the maximum of Yobs.
- Data is obtained from the kernel and thus excluded regions are ignored as shown in the plot above.
- Tabs for Cumulative Chi2 has been included in the appropriate GUI tabs as seen in the following:

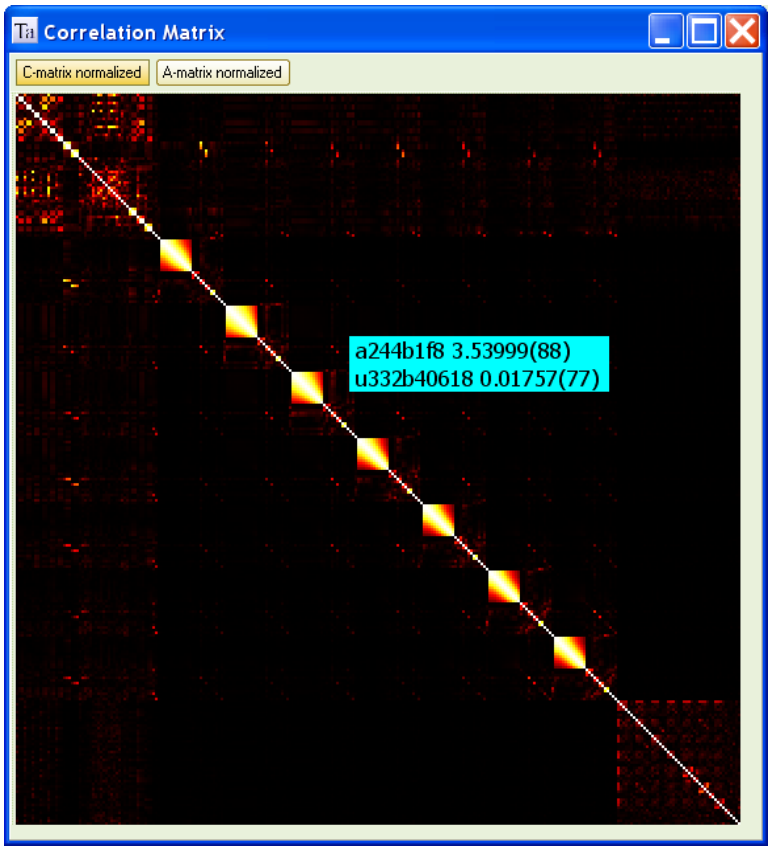


3.5 Correlation Matrix display

A Correlation matrix window activated from the Fit Dialog; it operates in Launch or GUI modes. Example output is as follows:



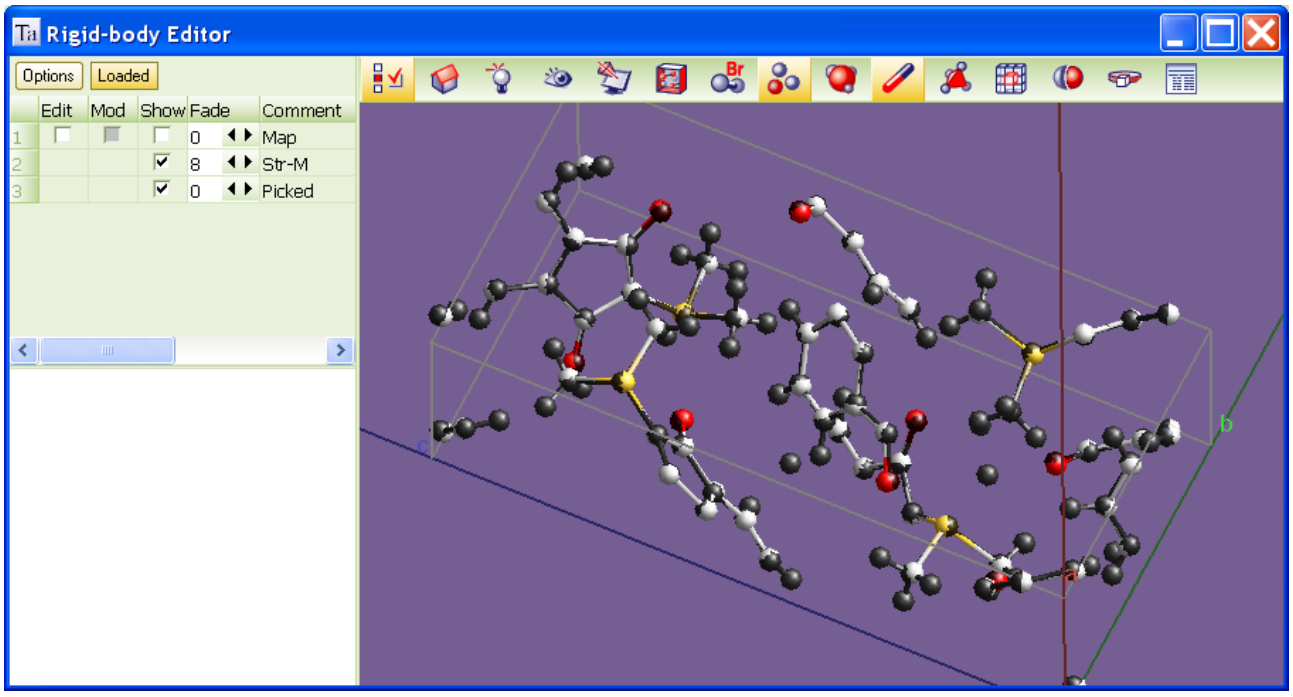
Both the A-matrix and the correlation matrix include penalties/restraints depending on whether *do_errors_include_penalties* and/or *do_errors_include_restraints* are defined. The display of the matrix can be zoomed using Ctrl-MouseWheel, for example:



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.

3.6 Fading a structure

The intensity of atom colours displayed in OpenGL can be adjusted using the Fade spin button of the OpenGL options grid; for example:

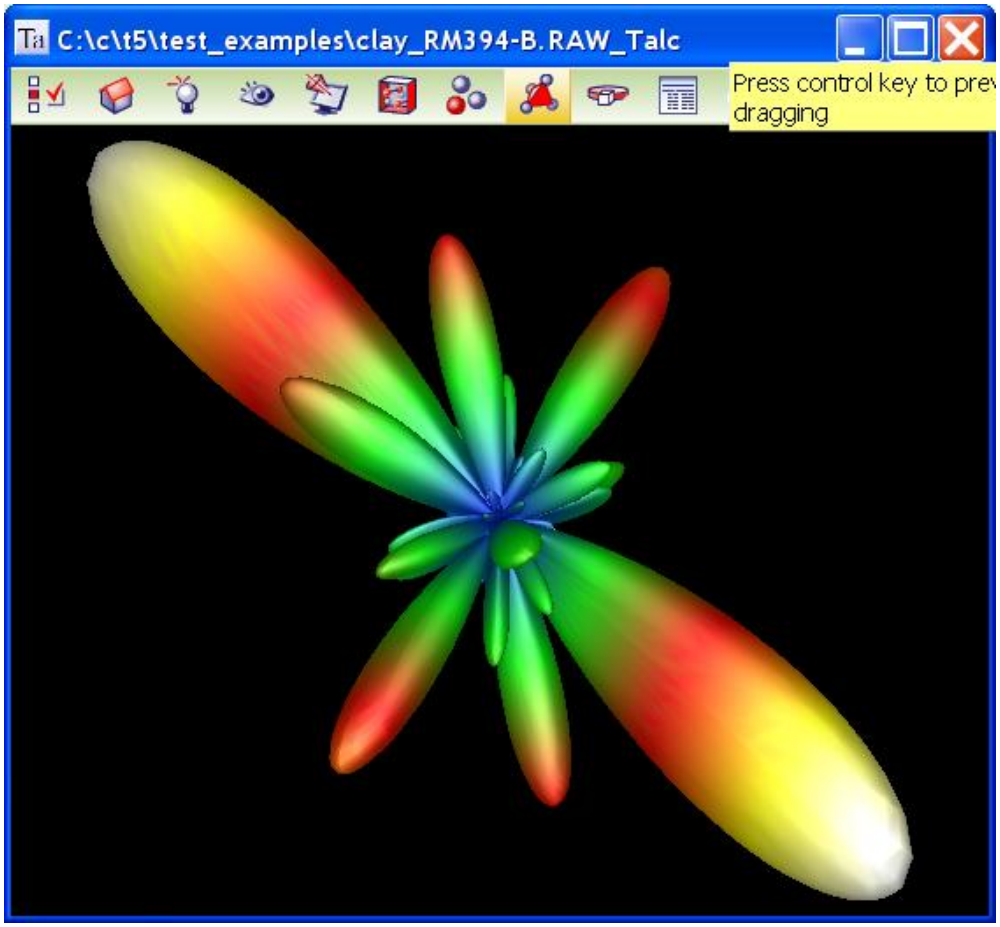


3.7 Normals Plot

An OpenGL plot of lattice plane Normals with the lengths of the Normals defined by the keyword *normals_plot*. For example:

```
normals_plot = Abs(H * K + L^2) + 1;
       normals_plot_min_d .3
```

normals_plot_min_d is optional; small values (ie. 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:



4 tc-inps.bat

The batch file TC-INPS.BAT found in the main TOPAS directory runs TC.EXE through around 80 test examples. Here's a snippet from TC-INPS.BAT

```
tc test_examples\mag\mag "#define CREATE_"
tc test_examples\mag\mag
tc test_examples\occ-merge "macro aac$ { iters 3000 }"
```

The macro called "aac$" instructs the program to place what's in aac$ at the bottom of the INP file. TC-INPS.BAT takes 3 to 5 minutes to run. Output can be placed into a file as follows:

```
tc-inps.bat > some_file
```

some_file will contain around 4000 lines of output.

5 Interface Specific

5.1 Improvements to the Grid

- Data can now be sorted by Double Clicking on Column Headings. Sorting alternates between ascending and descending order. On leaving a particular grid the column most recently sorted is remembered. On re-entry of that particular 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; ie. 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.

5.2 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.

6 Kernel Specific

6.1 New keywords

```
chi2                                mag_space_group
chk_for_best                        mg
current_peak_max_x                  mlx
current_peak_min_x                  mly
def                                 mlz
```

```
del_approx
do_errors_include_restraints
do_errors_include_penalties
element_weight_percent
element_weight_percent_known
elemental_composition
existing_prm
f0
f0_f1_f11_atom
f1
f11
ft_conv
ft_x_axis_range
ft_min
fit_obj_phase
fo_transform_X
fn
generate_these
generate_name_append
gui_add_bkg
layer
lpsd_beam_spill_correct_intensity
mag_atom_out
mag_only
mag_only_for_mag_sites
tangent_tiny
xdd_sum
modify_peak
modify_peak_apply_before_convolutions
modify_peak_eqn
more_accurate_Voigt
no_f11
no_inline
normals_plot
normals_plot_min_d
numerical_lor_gauss_conv
numerical_lor_ymin_on_ymax
out_prm_vals_dependents_filter
out_refinement_stats
remove_phase
report_on_str
return
save_best_chi2
scale_phase_X
stack
sx
sy
sz
user_y
WPPM_ft_conv
WPPM_L_max
WPPM_th2_range
WPPM_correct_Is
```

6.2 New Test Examples

test_examples\wppm\

compare-1.inp	**Launch**
cube-ln-normal-1.inp	**Launch**
gamma-fit-obj.inp	**Launch**
gamma.inp	**Launch**
ln-normal-1.inp	**Launch**
sphere-fit-obj.inp	**Launch**
sphere-gamma-compare-1.inp	**Launch**
sphere-gamma-compare-2.inp	**Launch**
sphere-gamma-compare-3.inp	**Launch**
super-lorentzian.inp	**Launch**
s-sphere-1.inp	**Launch**

test_examples\ft\

create-voigt.inp	**Launch**
alvo4a.inp	**Launch**
gaussian.inp	**Launch**
lorentzian.inp	**Launch**
voigt.inp	**Launch**

test_examples\single-crystal\

ylidma.inp	**Launch**
gebaa.inp	**Launch**
ae1-adps.inp	**Launch**
ae1-auto.inp	**Launch**
ae1-manual.inp	**Launch**
ae1-approx-a.inp	**Launch**
ae5-auto.inp	**Launch**
ae14-approx-a.inp	**Launch**
pn_02_2.inp	**Launch**

test_examples\tof\
- tof_balzar_br1.inp **Launch**
- tof_balzar_sh1.inp **Launch**
- tof_bank2_1.inp **Launch**
- tof_bank2_2.inp **Launch**

test_examples\user_y\
- cpd1e.inp **Launch**
- user_y.inp **Launch**

test_examples\lp-search\
- lp-search-cimetidine.pro **GUI**
- lp-search-pbso4.pro **GUI**
- lp-search-pbso4.inp **GUI-Launch**

test_examples\absorption-edge\
- Ni-LaB6.inp **GUI-Launch**
- Al2O3-pam.inp **GUI-Launch**
- Spinel-pam.inp **GUI-Launch**
- Rutile-Anatase.inp **Launch**
- Rutile-Anatase-Ni.inp **GUI-Launch**

test_examples\voigt-approx\
- create.inp **Launch**
- fit-more.inp **Launch**
- fit-pv.inp **Launch**
- fit-obj.inp **Launch**

test_examples\f0-f1-f11\
- xray-powder.inp **Launch**
- tof.inp **Launch**

test_examples\penalties-restraints\
- rosenbrock-10.inp **Launch**
- rosenbrock-10-restraint.inp **Launch**
- hock.inp **Launch**
- rastrigin.inp **Launch**

test_examples\mag\
- mag.inp **Launch**
- mag-2.inp **Launch**
- occ-merge.inp **Launch**
- mag-only.inp **Launch**
- maglamno3_magnetic.inp **Launch**

test_examples\quant\
- quant-1.inp **Launch**
- quant-2.inp **Launch**
- quant-3.inp **Launch**
- quant-4.inp **Launch**
- quant-5.inp **Launch**
- quant-6.inp **Launch**
- quant-7.inp **Launch**
- quant-7-create.inp **Launch**
- quant-8.inp
- zro2-restraint-wt.inp
- zro2-restraint-xrf-zr.inp

test_examples\rigid\
- rigidb.inp **Launch**
- rigida-1.inp **Launch**
- rigida-2.inp **Launch**

test_examples\rigid-errors\
- aniline_I_100K_x.inp **Launch**
- Aniline_I_8kbar_n.inp **Launch**

test_examples\stacking-faults\
- kaolinite-layer.inp **Launch**

debye-old.inp **Launch**
debye-new.inp **Launch**

test_examples\functions\

alvo4-fn.inp **Launch**
alvo4-normal.inp **Launch**
fn-test.inp **Launch**

test_examples\laue\

laue.inp **Launch**

test_examples\k-factor\

k-factor.inp **GUI-Launch**
k-factor.pro **GUI**

test_examples\

bkg-straight-line.inp **Launch**
chi2-ceo2.inp **Launch**
hash_prm.inp **Launch**
more_accurate_Voigt.inp **Launch**
occ-constrain.inp **Launch**
out_prm_vals.inp **Launch**
robust.inp **Launch**
scale_phase_X.inp **Launch**

6.3 New Equation Functions

```
Cosh
Erf_Approx
Erfc_Approx
Error
Gamma_Ln_Approx
Gamma_Approx
Get_Element_Weight
Ln_Normal_x_at_CD
Obj_There
Prm_There
Round
Sinh
Tanh
```

Example output from Round:

```
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
```

7 Pre-Processor

7.1 New Macros

```
Bkg_Straight_Line
Cu6_Ni_Edge
EP_Absorption_Edge_Correction
EP_Absorption_Edge_Correction_Eqn
LP_Factor_X
MM_Cartesian_Display
MM_Cartesian_Refine
Out_CIF_Bonds_Angles
Remove_Phase
Robust_Refinement
```

7.2**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 without the worry of name clashes. In the example:

```
macro Some_macro(b) { prm #m_unique a = Cos(Th); }
```

'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
{
    ' Rescale peaks according to 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
}
```

7.3 **Superfluous parentheses and the '&' Type for macros and its arguments**

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 there can be drawbacks; 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 the & Type can be used for macros that expect equation type arguments. Defining the macro with '&' before the arguments as in:

```
macro divide(& a, & b) { a / b }
prm e = divide(a+b, c-d);
```

instructs the pre-processor that the argument is of an equation Type and a check is made to determine whether the argument needs parentheses. This results in the correct expansion of:

```
prm e = (a+b) / (c-d);
```

Even with & types used 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 as follows:

```
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 now be written with less parentheses as follows:

```
macro & Ramp(& x1,& x2,& n)
{
      x1 + (x2-x1) Mod(Cycle_Iter, n) / (n-1)
}
```

7.4 **Pre-processor equations and #prm, #if, #elseif, #out, #m_if, #m_elseif, #m_out**

Pre-processor parameters, called hash parameters, can be defined by placing a # before the text prm. #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.

The #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:

```
#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
```

8 Keywords removed

```
swap_sites
try_site_patterns
break_if_been_there
hkl_Is_from_hkl4
do_processes
```

9 References

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; Evans, J.; Evans, I; Kern, A.; Parsons, S. (2011). Powder Diffraction, Vol. 26 Number 4 sup, *"The TOPAS symbolic computation system"*

David, W.I.F; Matteo, L.; Scardi, P. (2010). *Materials Science Forum Vol. 651 pp 187-200*

Leoni, M.; Di Maggio, R.; Polizzi, S; Scardi P. (2004), *J. Am. Ceram. Soc.* **87**, 1133-1140.

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"

Scardi, P. & Leoni, M. (2001). *Acta Cryst. A* **57**, 604-613.