TOPAS 64, Version 6

What's New

by Alan A. Coelho

September 8, 2016

64 bit address space for GUI and command line executables

Works with 64 bit Windows 7, 8 and 10

Multithreaded

Stacking faults at speed

PDF refinement at speed

Charge-Flipping for neutron data

Surface Plots

Macro enhancements

Contents

1 NEW KERNEL FUNCTIONALITY

1.1 Introducion

Directory test-mag have been removed to reduce file distribution size.

1.2 Miscellaneous

1.2.1 TC-INPS.BAT and the aac$ macro

The bath file TC-INPS.BAT runs through over 150 test examples in a time of a few minutes. These examples play an important role in program testing. Arguments passed via the command line to the test examples can contain the aac$ macro; if defined aac$ is expanded at the bottom of the INP file. For example, to terminate refinement after 100 iterations the following could be used:

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

1.2.2 TOPAS is now 64 bit

Version 6 utilizes 64 bit adressing. The command line TC.EXE and the GUI TA.EXE both run on the Windows 64 bit operating system. It means that all available memory can be used. The 64 bit compile has resulted in a 10 to 20% increase in execution speed.

1.2.2.1 Limiting Memory Usage – MaxMem.TXT

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

1) If A=0 then the maximum allowed memory usage becomes:

```
Max_Mem_Allowed = Max_Physical_Memory * B
```

In other words 75% of the total physical memory

2) If the number in MaxMem.TXT is negative then the maximum allowed memory usage becomes:

```
Max_Mem_Allowed = Max_Physical_Memory + A
```

3) If the number in MaxMem.TXT is positive then the maximum allowed memory usage becomes:

```
Max_Mem_Allowed = A
```

The default value in MaxMem.TXT is zero which corresponds to case (1). For all cases, if memory usage exceeds Max_Mem_Allowed then TC.EXE aborts with the message “Memory allocation limit reached, increase limit in file MaxMem.TXT”. TA.EXE aborts without a message; instead it creates the empty file called MaxMem-CHK.TXT. Checking the time/date stamp of MaxMem-CHK.TXT reveals whether TA.EXE aborted due to excessive memory usage.

1.2.3 Indexing - Figure of merit

The figure of merit *M* used in indexing is as follows:

$$M = 1 \Big/ \left((1+N_{\text{uni}})\, d_{o,\min}^{2}\, (N_c / N_o) \sum_i \left| d_{o,i}^2 - d_{c,i}^2 \right| Q_i \right) \tag{1-1}$$

$$\text{where } \; Q_i = N_o\, w_i \Big/ \sum_j w_j$$

Where d_o and d_c are the observed and calculated d-spacings, N_o and N_c the number of observed and calculated lines used, N_{uni} the number of unindexed lines found and the summations are over the used observed indexing lines. Q_i is a weighting that assists in the determination of extinction subgroups where w_i could for example be the inverse of the error in the peak positions from a Pawley refinement (see indexing\MgIr\index.inp). The keyword *index_I* corresponds to w_i. The formulation of Q_i is such that with or without Q_i the figure of merit *M* is of the same order of magnitude. The reciprocal-space lattice relationship solved during the indexing process (Coelho, 2000) includes *Q* as follows:

$$\begin{aligned} &(X_{hh}\, h^2 + X_{kk}\, k^2 + X_{ll}\, l^2 + X_{hk}\, h\, k + X_{hl}\, h\, l + X_{kl}\, k\, l \\ &\quad + Z_e\, (\pi/360)\, (4/\lambda^2) \sin(2\theta))\, W_{hkl} = W_{hkl} \,/\, d_o^2 \end{aligned} \tag{1-2}$$

$$\text{where } W_{hkl} = Q_{hkl}\, d_o^m \left| \Delta 2\theta_{hkl} \right|$$

1.2.3.1 Extinction subgroup determination

At the end of an indexing further indexing runs are internally performed across extinction subgroups in an attempt to determine the most likely subgroup. These internal runs are seeded with already determined lattice parameters and in most cases the correct extinction subgroup is obtained without the need for Q_i.

1.2.4 INP file enhancements Miscellaneous

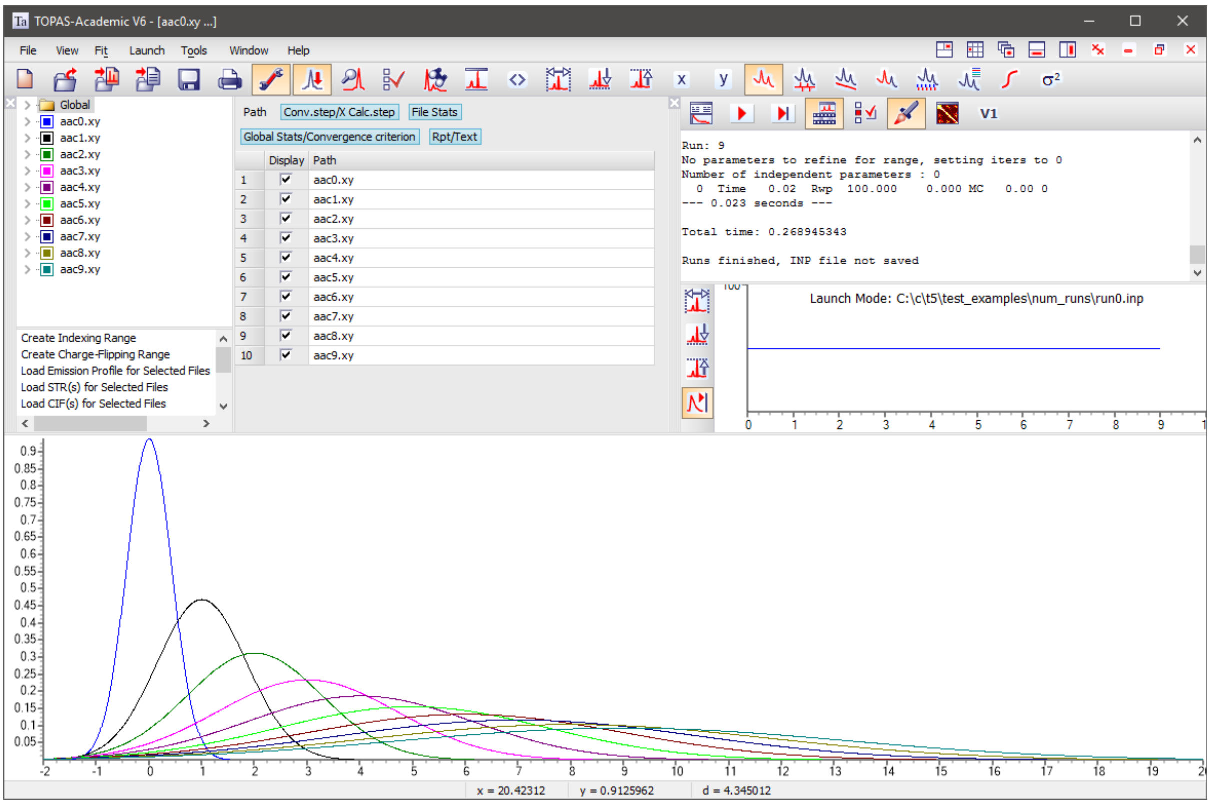
1.2.4.1 The num_runs keyword and Preprocessor improvements

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

num_runs defines the number of times the program executes (Runs) the INP file. Typically an INP file is run once; *num_runs* changes this behaviour where the refinement is restarted and then performed again until it is performed *num_runs* times. Information from one run to the next can be exchanged via the *out* keyword and the include keyword. The INP file is read each Run but is not updated when num_runs > 1 and out_file is empty. Equations during a Run could well simplify into a constant, or indeed, the Constant keyword can be used such that during a Run a parametr is not refined. From TB.EXE and Launch mode the Rwp graphical plot is appended such that it looks like *continue_after_convergence.* The following INP segment:

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

produces on execution the following:



TOPAS-Academic V6 - [aac0.xy ...]
File
View
Fit
Launch
Tools
Window
Help
Global
Path
Conv.step/X Calc.step
File Stats
Global Stats/Convergence criterion
Rpt/Text
Create Indexing Range
Create Charge-Flipping Range
Load Emission Profile for Selected Files
Load STR(s) for Selected Files
Load CIF(s) for Selected Files
V1
Launch Mode: C:\c\t5\test_examples\num_runs\run0.inp
x = 20.42312
y = 0.9125962
d = 4.345012

	Display	Path
1	✓	aac0.xy
2	✓	aac1.xy
3	✓	aac2.xy
4	✓	aac3.xy
5	✓	aac4.xy
6	✓	aac5.xy
7	✓	aac6.xy
8	✓	aac7.xy
9	✓	aac8.xy
10	✓	aac9.xy

```
Run: 9
No parameters to refine for range, setting iters to 0
Number of independent parameters : 0
  0  Time   0.02  Rwp  100.000    0.000 MC   0.00 0
--- 0.023 seconds ---

Total time: 0.268945343

Runs finished, INP file not saved
```

out_file detemines the name of the output file created when refinement terminates. The OUT file comprises the INP file but with prameter values updated. *out_file* defaults to the name of the INP file but with an OUT extension. If *num_runs* is greater than 1 and *out_file* is not defined then no OUT file is saved. This can speed up certain refinements where an OUT file is not needed. *out_file* is an equation that needs to evaluate to a string; here are som eexamples:

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

The standard macro Save_Best uses *out_file* as follows:

```
macro Save_Best {
      #if (Run_Number == 0)
```

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

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

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

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

1.2.4.2 Reserved macro Names

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

ROOT: Returns the root directory of the program.

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

Run_Number: Returns the current run numer.

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

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

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

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

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

1.2.4.3 The #list directive – creating arrays of macros

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

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

An element of the array is invoked using the first argument of the macro. In the case of File_Name and Time the first argument is implied. In the case of Temperature the first argument is empty as it needs to be. When the macro is invoked the first argument is a # type equation that must equate to an integer. Here's an example use of the File_Name macro in the above list:

```
xdd File_Name(Run_Number)
```

Curly brackets (as seen in the above list) can be used as delimiters within #list. The following:

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

will produce on expansion:

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

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

1.2.4.4 The File_Variable and File_Variables macro

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

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

Using File_Variable as follows:

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

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

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

will generate 11 *.XY files each generated with a different occupancy for the Ce1 site as detemined by the occ #prm. The names of the files would be occ0.xy to occ10.xy. Additionally, using the *system_after_save_OUT* keyword the file AAC.OUT will contain a concatenation of all the *.OUT files.

To iterate over two variable, pa and pb say, then the File_Variables macro, defined in TOPAS.INC as:

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

can be used as follows:

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

On running the above the PAPB.TXT file contains:

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

1.2.4.5 Equation String, Concat and Variable_Name_From_String functions

A distinction is made between parameter names and strings. Text occuring between double quotation marks or between brackets in the String function are deemed as String types. Note, the output from the following:

```
prm aabb = 1.234;
prm sasb = 4.321;
prm sa = "aa";
prm sb = "bb";
prm sc = sb;
prm = Variable_Name_From_String(Concat(sa, sc)); :  1.23400`
prm = Variable_Name_From_String(Concat(String(sa), String(sb))); :  4.32100`
```

The arguments of Concat are by default variable names and not strings. Here are some examples:

```
prm = Concat("a", "b", "c"); ' result = "abc"
prm = Concat(String(a), "b", "c"); ' result = "abc"
prm p = "a"; prm = Concat(p, "b", "c"); 'result = "abc"
```

The Save_Best macro uses strings as follows:

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

1.2.4.6 dummy and dummy_prm keywords

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

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

1.2.5 out_dependences and out_dependences_for

```
out_dependences $user_string
out_dependences_for $user_string $object_name
```

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

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

produces on refinement termination the following in standard output (fit window or DOS command line):

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

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

```
rigid
     …
     out_dependents_for tag_1 rigid
```

There are many $object_name's that are valid. Basically all parameters can be tagged, e.g.

```
x, y, z, occ, beq, u11, u22, u33, u12, u13, u23, a, b, c, al, be, ga, etc…
```

In addition the following non-parameters can be tagged:

```
site, rigid, sites_restrain, lat_prms, gauss_conv,  lor_conv,  all_scale_pks,
th2_offset_eqn
```

1.2.6 The peaks buffer, speed and memory considerations

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

```
[str...] [hkl_Is...] [xo_Is...] [d_Is...]
   peak_buffer_based_on !E...
       peak_buffer_based_on_tol !E
```

The normal means of determining the size of the peak buffer is over ruled when *peak_buffer_based_on* is defined. With *peak_buffer_based_on* peaks are grouped accoding to the *peak_buffer_based_on* criterion. For example, to insert a peak into the peak buffer at x-axis intervals of 1 then the following can be used:

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

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

```
    peak_buffer_based_on = Xo;
       peak_buffer_based_on_tol .01
  peak_buffer_based_on = sh;
       peak_buffer_based_on_tol 1e-7
```

Where *sh* can be a spherical harmincs parameter or an equation describing hkl dependence or a *march_dollase* parameter. When more that one *peak_buffer_based_on* is defined then peak groups formed obey all of the *peak_buffer_based_on*'s.

peak_buffer_based_on disables the peak stretching procedures and any defined *aberration_range_change_allowed*. *peak_buffer_based_on* can be a function of the reserved parameters H, K, L, M, D_spacing, X, Xo, Th.

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

```
peak_buffer_based_on 1
    peak_buffer_based_on_tol -1
```

1.2.7 Threading

TOPAS is threaded to a large extent; this allows the utilization of multiple processors which results in faster program execution. The degree of speedup is computer and problem dependent. For nontrivial problems the gain is 2 to 4 for a 4 core laptop PC with four i7 processors as seen in the following table.

			Time (secs)		Gain	
INP File	Comment	V6	V5	V4	V4/V6	V5/V6
\ft\alvo4a "#define TEST_ #define USE_FT_ #define USE_SH_"	ft, spherical harmonics	1.70	6.75	-	-	3.97
\absorption-edge\Al2O3-pam	Absorption edge, modify peak	1.46	5.67	-	-	3.87
\cime-z-auto "#define TEST_2_"	no decompose	7.08	25.03	18.70	2.64	3.54
\tof\tof_bank2_2	user_defined_convolution	0.18	0.63	0.53	2.93	3.49
\wppm\cube-ln-normal-1 "#define TEST_"		2.34	8.05	-	-	3.44
\alvo4a "#define TEST_"		1.22	3.95	5.52	4.52	3.24
\single-crystal\ae14-approx-a "#define TEST_"		1.03	3.31	11.47	11.14	3.21
\mag\mag "#define TEST_"		1.09	3.20	-	-	2.94
\single-crystal\pn_02_2-adps	3970 parameters	6.32	17.37	27.88	4.41	2.75
\single-crystal\ae14-adps		0.29	0.78	1.27	4.46	2.73
\li250		0.39	1.06	0.81	2.08	2.72
\tube-tails "#define TEST_"		1.48	3.80	3.20	2.16	2.56
\pbso4a "#define TEST_"		2.29	5.74	6.42	2.80	2.50
\single-crystal\pn_02_2 "#define TEST_1_"		3.23	8.01	11.73	3.63	2.48
\sp\serine_I_evans_N_ta_bang_rot-z	z-matrix	0.67	1.62	2.35	3.51	2.41
\ft\alvo4a "#define TEST_ #define USE_FT_"	ft, no spherical harmonics	0.38	0.91	-	-	2.38
\ft\alvo4a "#define TEST_"	no ft, no spherical harmonics	0.21	0.48	-	-	2.29
\single-crystal\ae14-adps "#define TEST_"	no approximate_A	0.57	1.31	5.74	10.04	2.29
\capillary-lpsd\capillary-simulated		0.10	0.22	-	-	2.28
\y2o3a		0.24	0.54	0.44	1.82	2.25
\peak-intensity-extraction\zhu3lebail		0.16	0.34	0.49	3.14	2.20
\sp\serine_I_evans_N_ta_bang_rot-z		0.82	1.79	2.28	2.78	2.18
\Voigt-approx\fit-obj.inp "#define TEST_"		4.39	9.55	-	-	2.18
\zro2 "#define TEST_"		1.87	3.82	-	-	2.05
\cime-z-auto "#define TEST_1_"	decompose	6.95	13.87	15.20	2.19	2.00
\k-factor\k-factor		0.33	0.64	-	-	1.95
\single-crystal\pn_02_2 "#define TEST_2_"	no approximate_A	25.68	49.30	97.15	3.78	1.92
\mag\JohnEvans\jsoe_fit_p1_tofullproftric_01 "macro aac$ { iters 30 }"		4.00	7.61	-	-	1.90
\peak-intensity-extraction\pawley1 "#define TEST_"		1.73	3.16	3.98	2.30	1.83
\cf\cf-1a7y “#define TEST_”	Charge Flipping	12.87	23.40	25.81	2.00	1.81
\ft\alvo4a "#define TEST_ #define USE_SH_"	no ft, spherical harmonics	0.53	0.95	-	-	1.79
\sp\serine_i_evans_n_ta_bang_rot		0.74	1.27	1.90	2.57	1.71
\pvs "#define TEST_"		1.22	2.08	4.94	4.05	1.70
\stacking-faults\kaolinite "#define TEST_1_ #define Speed"	with Speed	0.58	0.97	-	-	1.66
\stacking-faults\kaolinite "#define TEST_1_"		3.10	4.96	-	-	1.60
\clay		0.75	1.15	2.90	3.87	1.53
\tof\tof_bank2_1 "#define TEST_"		0.60	0.92	2.28	3.80	1.53

\occ-merge "#define TEST_"		3.37	4.90	6.20	1.84	1.45
\simon\alan	Many smalll strs	0.73	0.97	3.36	4.60	1.33

Attention has been paid to reducing memory usage at the thread level. This is particularly apparent when using rigid bodies or occupancy merge where Version 6 uses far less memory than Version 5. Except for penalties all items are threaded and they include:

- Peak generation
- All colvolutions
- All derivatives that are a function of Ycalc
- Equations that are a function of changing variables such as X, Th, D_spacing etc...
- Pawley refinement
- Strucure refienment
- Charge Flipping
- Magnetic refinement
- Stacking faults
- PDF refinement
- Conjugate gradient solution method
- Indexing

1.2.7.1 Setting the maximum number of threads - MaxNumThreads.TXT

The maximum number threads available is used by default. The user can limit the maximum number of threads by editing the file MaxNumThreads.TXT. This file is read on porgram startup; it contains a single number, lets call it Max_Threads_File, which defines the maximum number of threads. Non-existance of the file or a Max_Threads_File of zero results in the porgram using the maximum number of threads available. If Max_Threads_File is negative then the maximum number of threads is set to the following:

Max_Number_Threads = Max(1, Max_Threads_File + Max_Threads_Available);

1.2.8 Using local to assist in using "for … {}" loops

The following parameters have had their status changed from global to 'local' allowing their use in 'for' loops.

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

To constrain the march_dollase parameter, as used in the PO macro, to the same value within a "for xdds { for strs { … }}" construct across two or more structures then simply give them the same name, for example:

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

See examples po-constrained-create.inp and po-for.inp in the test_examples\po-constrained directory. Note also the use of "if Prm_Then(…) { … }" rather than "for strs 1 to 1 {…}" to facilitate the writing of the INP file.

The $Name in spherical_harmonics_hkl is 'local' but the spherical harmonics coefficients are global. In the following:

```
PO_Spherical_Harmonics(sh2, 8 load sh_Cij_prm {
      k00   !sh2_c00  1.0000
      k41   sh2_c41   0.1000
      k61   sh2_c61  -0.2000
      k62   sh2_c62   0.3000
      k81   sh2_c81  -0.4000 } )
```

the sh2 parameter is local to the str and the coefficients k00, k41 etc… are global. This allows the constraining of coefficients across different structures within "for strs"; see examples posh-constrained-create.inp and posh-for.inp in the test_examples\po-constrained directory:

1.2.9 Charge Flipping and neutron_data

The *neutron_data* keyword informs the charge flipping routine that neutron scattering lengths are to be used. It also results in the following default neutron flipping routine being used:

```
flip_equation =
      If(And(Get(density) < Get(threshold), Get(density) > 0.4
      Get(min_density)),
            -Get(density),
             Get(density)
      );
```

The *flip_neutron* equation defines the 0.4 in the above equation. For example, the following can be used to change the default to 0.5:

```
flip_neutron = 0.5;
```

The tangent formula is made less accurate due to the negative scattering of H atoms. However, if positive scattering lengths are dominant then the tangent formula can stabilize refinement. For example, try:

```
Tangent(.3, 30)
      tangent_scale_difference_by = Ramp(1, 0, Nc);
```

See test_examples\cf\neutron-cime\cf-neutron.inp

1.2.9.1 Powder data, the A matrix and the Tangent formula

In the case of charge-flipping from powder data *TOPAS* uses the diagonally normalized A-matrix *cf_in_A_matrix* (see example cf\cf-cime.inp)**,** which we will call **D,** from a Pawley refinement (see example cf\cf-cime-pawley.inp) to modify normalized structure factors $\mathbf{E}_h$ calculated during the charge-flipping process; this produces better results than using reflections output in a SHELX format. Equation (1-3) shows how the structure factors are modified.

$$\mathbf{E}_{h,calc,\text{modified}} = \mathbf{E}_{h,calc}\sqrt{\frac{I_{obs,h,w}}{I_{calc,h,w}}} \tag{1-3}$$

where $I_{calc,h,w} = \sum_k \mathrm{D}_{h,k}^2\, I_{calc,k}$ and $I_{obs,h,w} = \sum_k \mathrm{D}_{h,k}^2\, I_{obs,k}$

The subscripts *h* and *k* corresponds to reflections *h* and *k* respectively and the summation in *k* is over all reflections. $I_{calc,k}$ and $I_{obs,k}$ corresponds to observed and calculated intensities. Equation (1) modifies the calculated intensities to include intensities from overlapping peaks. When there's no overlap $D_{i,i}$=1 and $D_{i,j}$=0 and the calculated intensities as well as $\mathbf{E}_h$ are not modified. When using the direct-methods tangent formula within the charge-flipping process as described by Coelho (2007), the **D** matrix is also used to modify E_h values used in triple phase relationships as shown in equation (2).

$$E_{obs,h,\text{modified}} = E_{obs,h}\, q_h + E_{calc,h}\,(1-q_h) \tag{1-4}$$

where $q_h = \sum_k \mathrm{D}_{h,k}^2$

$E_{obs,h}$ and $E_{calc,h}$ corresponds to tangent formula E_h values calculated from the observed and calculated intensities respectively. $E_{calc,h}$ is typically not used in the tangent formula, however, intensities used for determining $E_{obs,h}$ can be grossly in error due to peak overlap. Equation (2) therefore influences triple phase relationships by weighing $E_{obs,h}$ by q_h; when there's no overlap q_h=1 resulting in no modification. When there's significant overlap then q_h is small and the influence of triple phase relationships using the *h* reflection is reduced. Equation (2) also includes a (1- q_h) portion of $E_{calc,h}$ thus stating that when there's significant overlap the calculated E_h is to be more trustworthy than the observed E_h. Equation (1-4) corrects for errors in E_h when I_{obs} is similar to I_{calc}; this assists in reducing the goodness of fit value thus enhancing the chances of solving the structure.

1.2.10 Error determination using SVD

Errors have previously been determined from a covariance matrix obtained by LU decomposition. Version 6 uses Singular Value Decomposition (SVD) with resulting errors typically smaller for strongly correlated parameters. Additionally SVD errors more closely resemble those obtained by the boot strap method. *bootstrap_errors* are potentially more accurate as it considers parameter limits; for example the fact that intensities are positive is not considered by matrix inversion.

The keyword *use_LU_for_errors* can be used to force the use of LU decomposition. The three means of determining errors are demonstrated in a Pawley refinement of Y2O23 in the following INP files:

```
Test_examples\svd-errors\y2o3a-lu.inp
Test_examples\svd-errors\y2o3a-svd.inp
Test_examples\svd-errors\y2o3a-boot.inp
```

Comparisons showing the similarity between the bootstrap and SVD errors are seen in the following INP snippets:

```
LU    Full_Axial_Model(12, 15, 12, 5.1, @  8.56191`_0.06668)
SVD   Full_Axial_Model(12, 15, 12, 5.1, @  8.56190`_0.05045)
Boot  Full_Axial_Model(12, 15, 12, 5.1, @  8.56191`_0.04988)
```

```
LU    ZE(@, 0.08739`_0.00037)
SVD   ZE(@, 0.08739`_0.00018)
Boot  ZE(@, 0.08739`_0.00020)

LU    Cubic(@  10.605643`_0.000013)
SVD   Cubic(@  10.605643`_0.000019)
Boot  Cubic(@  10.605643`_0.000020)

LU    CS_L(csl, 369.29260`_10.37269)
SVD   CS_L(csl, 369.29260`_3.13254)
Boot  CS_L(csl, 369.27962`_2.92401)

LU   bkg  @  199.402275`_21.4050278 -8.89848018`_1.85486669  47.9112026`_1.841 83168..
SVD  bkg  @  199.407893`_1.12005938 -8.91085058`_1.84031795  47.9154076`_1.69533286...
Boot bkg  @  199.402275`_0.92177244 -8.89848018`_1.47936429  47.9112026`_1.20647356...

                     LU                             SVD                  Boot
0   0   2 ... 1.02580`_0.12061              1.02578`_0.06440     1.02580`_0.04088
2   1   1 ... 65.81458`_0.43091             65.81449`_0.43023    65.81458`_0.33495
0   2   2 ... 0.00000`_0.07691              0.00003`_0.07645     0.00000`_0.00138
2   2   2 ... 1197.20758`_2.76303           1197.20695`_2.5964   1197.20758`_3.88388
3   2   1 ... 2.55249`_32911418.00252       2.55243`_0.12020     2.55249`_0.07342
2   3   1 ... 2.55249`_14161941.24442       2.55243`_0.12020     2.55249`_0.07342
0   0   4 ... 374.70367`_1.83981            374.70343`_1.71262   374.70367`_1.46988
4   1   1 ... 88.85977`_0.91217             88.85964`_0.91858    88.85977`_0.65040
4   0   2 ... 10.65522`_16487463.58725      10.65514`_0.26162    10.65522`_0.19312
0   4   2 ... 10.65522`_16487463.58725      10.65514`_0.26162    10.65522`_0.19312
3   3   2 ... 141.18328`_1.24215            141.18308`_1.28901   141.18328`_1.07037
4   2   2 ... 21.67021`_0.61141             21.67012`_0.59628    21.67021`_0.42625
3   4   1 ... 119.04993`_124057100727963620 119.04970`_0.91382   119.04993`_0.88560
4   3   1 ... 119.04993`_124057100727963630 119.04970`_0.91382   119.04993`_0.88560
5   2   1 ... 44.77524`_31163203794483688   44.77511`_0.63035    44.77524`_0.43513
2   5   1 ... 44.77524`_31163203794483724   44.77511`_0.63035    44.77524`_0.43513
0   4   4 ... 1558.50793`_5.05350           1558.50756`_5.1797   1558.50793`_7.02290
4   3   3 ... 91.53419`_1.30219             91.53401`_1.37722    91.53419`_0.94397
```

Note the very large errors obtained by LU-decomposition for intensities that are 100% correlated.

1.2.11 Error Propagation using prm_with_error

Fixed parameter errors determined outside of refinement can be included and propagated to dependent parameters using the keyword *prm_with_error*. For example, consider the INP snippet (see test_example\prm-with-error.inp):

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

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

1.2.12 New Keywords

Typically keywords can be placed anywhere within an allowed scope. Due to the increasing number of keywords Version 6 introduces delimited keyword blocks which at present only applies to the *generate_stack_sequences* block. Keywords that are dependents of *generate_stack_sequences* can only be used within the *generate_stack_sequences* block. This avoids keyword name clashes and allows for keywords to be more readable. For example, *generate_stack_sequences* has the depdnents of *ta*, *tb* and *tz* which are also keywords used by the *translate* keyword dependents. Use of these kewyords however is possible as *generate_stack_sequences* uses opening and closing curly brack delimiters; as follows:

```
generate_stack_sequences {
   ' generate_stack_sequences dependents placed here
   gauss_fwhm 0.02  ' non-generate_stack_sequences depndent
}
```

Non-dependent keywords of *generate_stack_sequences* can still be placed within the *generate_stack_sequences* block.

New Version 6 keywords are as follows:

```
a_add
b_add
flip_neutron
generate_stack_sequences
height
layers_tol
match_transition_matrix_stats
n
number_of_sequences
number_of_stacks_per_sequence
num_runs
num_unique_vx_vy
n_avg
out_dependences
out_dependents_for
out_file
pdf_convolute
pdf_data
pdf_scale_simple
peak_buffer_based_on
peak_buffer_based_on_tol
prm_with_error
save_sequences
save_sequences_as_strs
system_after_save_OUT
system_before_save_OUT
to
transition
use_layer
use_LU_for_errors
z_add
convolute_X_recal
pdf_for_pairs
pdf_gauss_fwhm
pdf_info
pdf_only_eq_0
pdf_ymin_on_ymax
```

```
        pdf_zero
```

1.2.13 New Functions

Gamma_P(a, x): Returns the incomplete Gamma function P(a, x)

Gamma_Q(a, x): Returns the incomplete Gamma function Q(a, x) = 1-P(a,x)

1.3 Stacking faults

```
[generate_stack_sequences] {
   [number_of_sequences !E]
   [number_of_stacks_per_sequence !E]
   [save_sequences $file]
   [save_sequences_as_strs $file]
   [layers_tol !E]
   [n_avg !E]
   [num_unique_vx_vy !N]
   [match_transition_matrix_stats { ... }]
   [transition $transition_name]...
      [use_layer $layer]
      [height E]
      [n !N]
      [to $to_transition_name !E]...
         [ta E] [tb E] [tz E]
         [a_add E] [b_add E] [z_add E]
}
' Get(generated_c)
```

Examples:

```
        test_examples\stacking_faults\
                fit-1.inp
                fit-2.inp
                fit-3.inp

        test_examples\stacking-faults\Rietveld-Generate\
                Rietveld-Generate\Create-Sequences.inp
                Rietveld-Generate\Rietveld-Generate.inp
                Rietveld-Generate\Fit-to-Rietveld-Generated.INP
                Rietveld-Generate\Rietveld-Generated-200-2000.xy
                Rietveld-Generate\strs-200-2000.txt
```

Stacking fault generation and refinement can now be performed at speeds that make routine analysis possible.

generate_stack_sequences generates sequences of stacks from the transition matrix described by the transition keywords. The opening and closing braces of {} corresponds to a block where keywords local to *generate_stack_sequences* can be used. Outside of the braces the *generate_stack_sequences* can't be used. Get(generated_c) is updated after generation of the sequences with the average thickness of the sequences. It can be used to set the *c* lattice parameter.

num_unique_vx_vy: On termination of refinement the number of unique { sx, sy } stacking vector coordinates is reported for all layer types.

transition: defines a 'from' transition with the name $transition_name. The transition uses the layer defined in *use_layer*.

to: defines the to-transition. $to_transition_name must be a defined $transition_name.

n returns the number of transitions generated for the corresponding *to* to-transition.

height: can be used instead of *z_add* keywords.

ta, *tb*: defines the stacking vector x and y coordinates in terms of the crystallographic ***a*** and ***b*** axes.

a_add, *b_add*: defines the stacking vector x and y coordinates relative to the previous stacking vector in terms of the crystallographic ***a*** and ***b*** axes.

tz: defines stacking vector z coordinate along the crystallographic ***c*** axis in Å.

add_z: defines stacking vector z coordinate along the crystallographic ***c*** axis in Å relative to the previous stacking vector.

1.3.1 Generating the same stacking sequences each run

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

```
seed #number
```

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

1.3.2 The SF_Smooth macro

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

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

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

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

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

1.3.3 Fitting to DIFFaX test diamond data

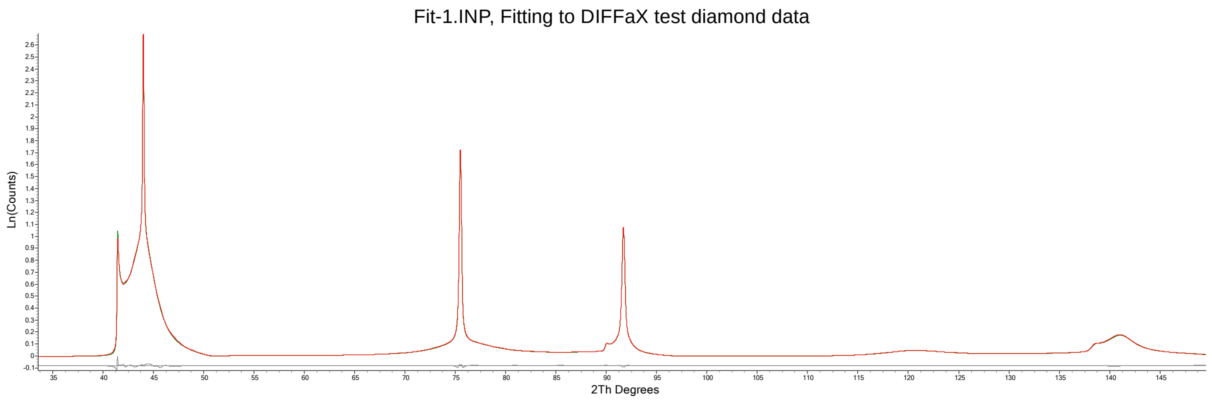
Fit-1.INP uses generate_stack_sequences to fit to data generated from the DIFFaX suite (Treacy, 1991); INP segment that generates the sequences looks like:

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

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

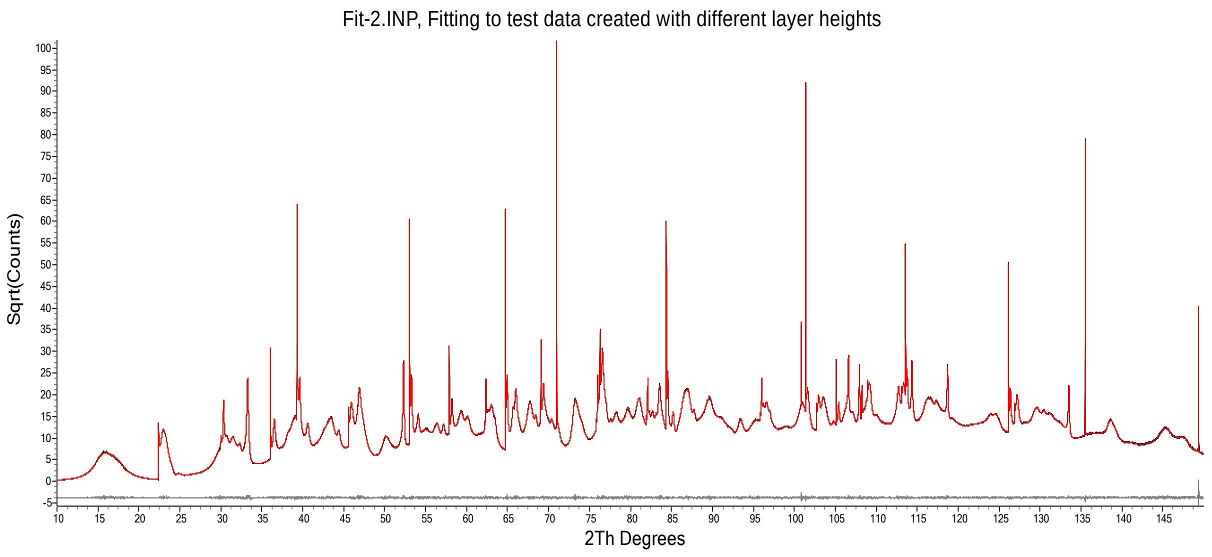
```
prm !pa_gen = (n1+n3)/(n1+n2+n3+n4); : 0.699974874
```

The fit to the DIFFaX data looks like:



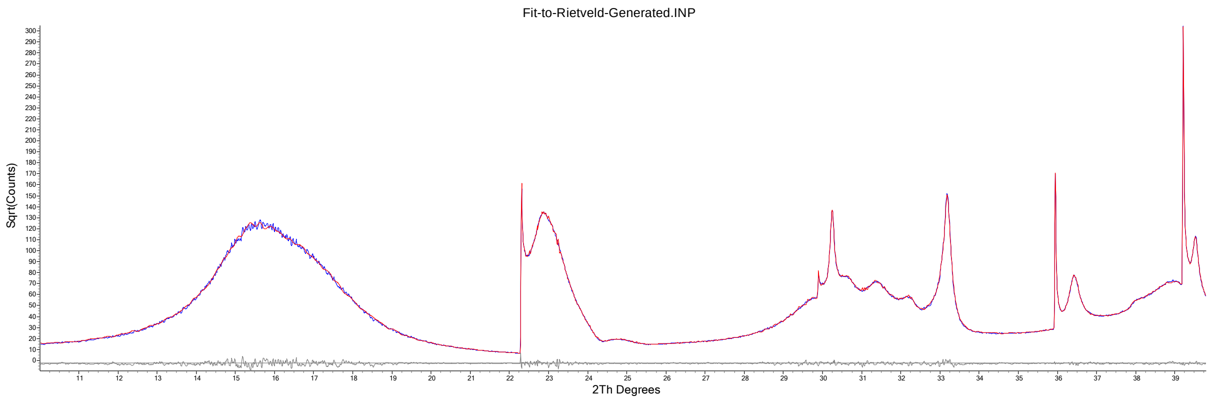
1.3.4 Stacking faults from layers of different layer heights

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



1.3.5 Rietveld-Generated example

The files in the test_examples\stacking-faults\Rietveld-Generate\ directory can be used to create a stacking faulted test pattern using Rietveld refinement. The test pattern can then be fitted to. Create-Sequences.inp created the INP format stacking sequences and places the result int the file strs-200-2000.txt. The file Rietveld-Generate.inp can then be used to create the test pattern Rietveld-Generated-200-2000.xy. This test pattern can be fitted to using the file Fit-to-Rietveld-Generated.INP; this file uses generate_stack_sequences. It demonstrates the accuracy and speed of the stacking fault averaging fitting procedure. The fit to the Rietveld generated stacking faulted pattern looks like:



1.3.6 Refining on layer heights

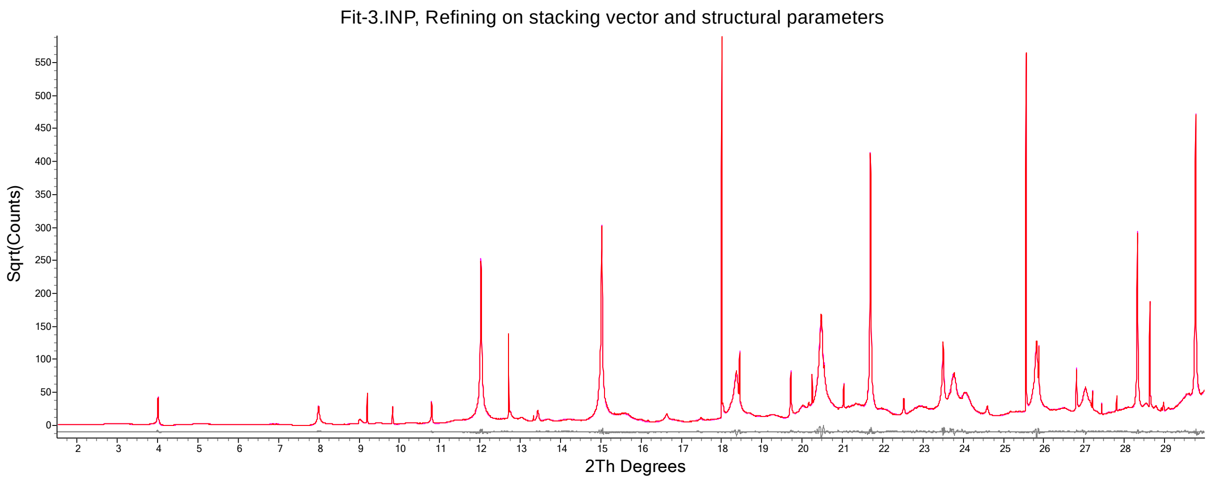
Layer heights can be refined by refining on parameters that are a function of the *add_z* or *height* keywords. The Fit-3.INP example refines on 3 height parameters as well as the z fractional atomic coordinates of the sites that comprise the layers. It also lists six types of transitions which operate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

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

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

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

The fit to the test data looks like:



1.4 PDF refinement

PDF patterns are treated as an '*xdd*'. Many keywords used for xdds can also be used in PDF refinement. PDF patterns can be refined simultaneously with other types of *xdd* patterns comprising x-ray dependent or x-ray independent phases or peaks phases. Penalties, restraints and in particular keywords such as *rigid*, *atomic_interaction*, *sites_geometry*, *sites_distance* etc… can all be used. Data structure items frequently used in PDF refinement are:

```
xdd…
   pdf_data
   *[scale_phase_X E]…
   *[fit_obj]…
    [start_X #] [finish_X #]
   *[rebin_with_dx_of !E] [rebin_start_x_at !E]
   *[weighting !E]
   *[Tpdf_convolute]…
   str…
      *[scale_phase_X E]…
       [scale E]
       [view_structure]
       [rigid]…
       [occ_merge $sites]…
       [pdf_scale_simple]
      *[pdf_zero E]
       [pdf_ymin_on_ymax 0.001]
       [pdf_info]
      *[Tpdf_convolute]…
       [pdf_for_pairs $sites_1 $sites_2]…
          [pdf_only_eq_0]
         *[pdf_gauss_fwhm E]
```

```
        *[Tpdf_convolute]…
```

where

```
      Tpdf_convolute
        *[pdf_convolute E]…
           [min_X !E]
           [max_X !E]
           [convolute_X_recal !E]
```

Examples files in directory test_examples\pdf

```
      beq-2.inp
      beq-2-create.inp
      beq-3.inp
      beq-3-create.inp
      pdf-1.inp
      pdf-2.inp
      alvo4\structure-solution-create.inp
      alvo4\structure-solution.inp
      alvo4\rigid.inp
      occ-merge-pbso4\create.inp
      occ-merge-pbso4\occ-merge-test.inp
      occ-merge-pbso4\occ-merge.inp
```

Data files in directory test_examples\pdf

```
      beq-2.xy
      beq-3.xy
      alvo4\alvo4.xy
      occ-merge-pbso4\pbso4.xy
```

Keywords with '*' next to them can be a function of the x-axis reserved parameter name X; for PDF data X corresponds to r.

pdf_data tells the program that the data set is of G(r) type.

Let's write G(r) as follows:

G(r) = s1 S(r) / r – s2 r

where

r corresponds to the X-axis

s1 and s2 are constants

S(r) are the pairs

pdf_scale_simple tells the program to calculate S(r)/(Np r) only.

pdf_ymin_on_ymax defines the minimum/maximum value for the Gaussians in regards to the x-axis range calculated for the Gaussians; the default value of 0.001 in typically sufficient.

pdf_for_pairs can be used to select site pairs using the site name; for example:

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

The '!' character excludes the sequence from the wild card string. Multiple *pdf_for_pairs* can be defined.

pdf_only_eq_0 informs the parent *pdf_for_pairs* that only equivalent position 0 is to be considered.

pdf_gauss_fwhm is used to write the width equation for the pairs selected by *pdf_for_pairs*. If all of the pairs possible are described by *pdf_for_pairs* then the associated *beq*'s are not used and they become redundant. The user is informed of unused *beq*'s. Consider the following abbreviated INP segment:

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

There are a number of types of interactions with FWHMs as follows:

```
Al1↔O1  interactions for equivalent position 0 described using Line B
Al1↔O1  interactions excluding equivalent position zero described using Line C
O1↔O1   interactions described using beq's
```

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

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

   Al1    -2
   O1     2-
}
```

The matrix is shown in blue. *pdf_for_pairs* together with *beq*'s defaults offers great flexibility in describing peak widths. See PDF-1.INP, PDF-2.INP, BEQ-3.INP.

scale_phase_X can be used to describe Gaussian dampening, for example:

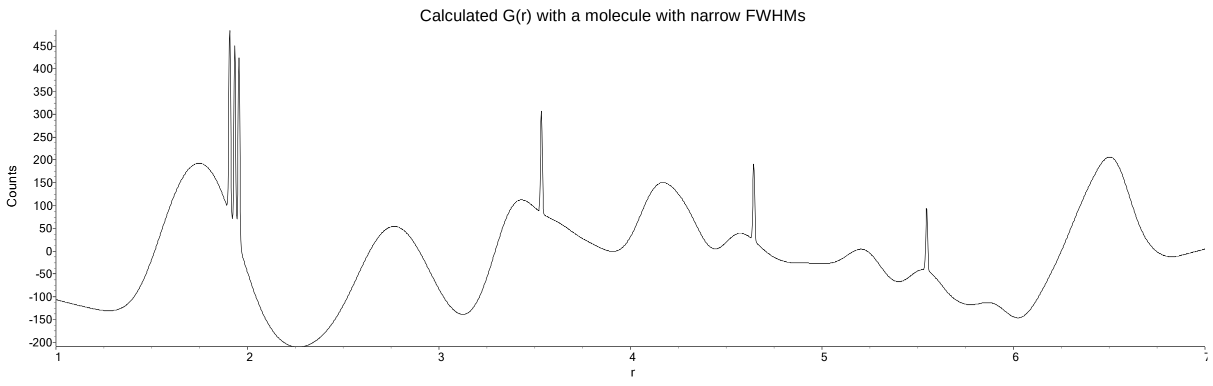
```
prm damp_fwhm  50 min 1e-6 max 200
prm damp = Gauss(0, damp_fwhm);
scale_phase_X = damp;
```

1.4.1 Inter and Intra molecule FWHMs

Assigning different interaction types for molecules is done using pdf_for_pairs. For example, to set the bond lengths for the atom Al1 of AlVO4 (see PDF-2.INP) to a different FWHM for equivalent position 0 the following could be used:

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

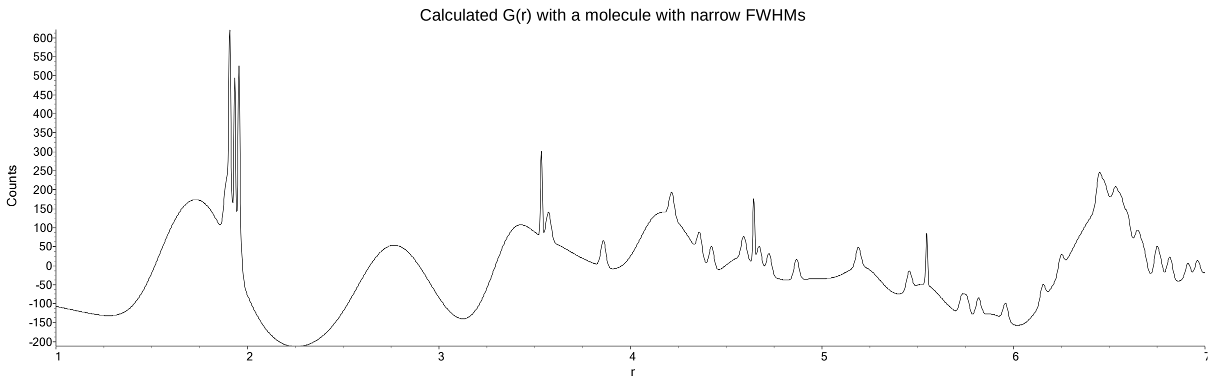
The calculated pattern from PDF-2.INP therefore becomes:



Notice the 6 spikes; they correspond to the Al1 bonds which has narrow FWHMs. If we then wanted Al1 bonds that are not equivalent position 0 to be different to the *beq*'s values then we could use:

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

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



The corresponding output from pdf_info is:

```
pdf_info
{
```

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

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

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

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

The following will not throw an exception:

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

1.4.2 Instrument Sinc function Sinc-1.INP

In Sinc-1.PDF *pdf_convolute* is used at the *xdd* level to convolute a Sinc function into pdf type phases as follows:

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

Sinc-1.INP also uses an xo_Is phase defined as:

```
xo_Is
   NoThDependence(0.0001)
   xo 10 I @ 100
   peak_type pv
```

```
            pv_lor 0.5
            pv_fwhm 2
```

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

1.4.3 Weighting of PDF and 2-Theta type data

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

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

1.4.4 Test_examples\pdf\BEQ-2.INP

Use Test_examples\pdf\BEQ-2-create.INP to generate a simulated pattern for BEQ-2.INP

BEQ-2.INP

- Comprise the structure of AlVO4
- 3 types of *beq* parameters
- beq is a function of X (ie. X corresponds to the X-axis which is r) and hence peak widths are a function of X.
- demonstrates the use of *pdf_zero*
- demonstrates the use of *rebin_with_dx_of* and *rebin_start_x_at*

1.4.5 Test_examples\pdf\BEQ-3.INP

Use Test_examples\pdf\BEQ-3-create.INP to generate a simulated pattern for BEQ-3.INP

BEQ-3.INP

- Demonstrates the use of *pdf_for_pairs*

1.4.6 Speeding up refinement with *rebin_with_dx_of*

The step size in PDF data must be of equal size. Also, the start of the x-axis needs to be an integral multiple of the step size. Increasing the step size in the data speeds up refinement; see BEQ-2.INP. The step size can be increased using:

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

Rebinning is akin to collecting the data at a larger step size. All data is included with counts after rebinning being equal to counts before rebinning. esds associated with the data are also rebinned.

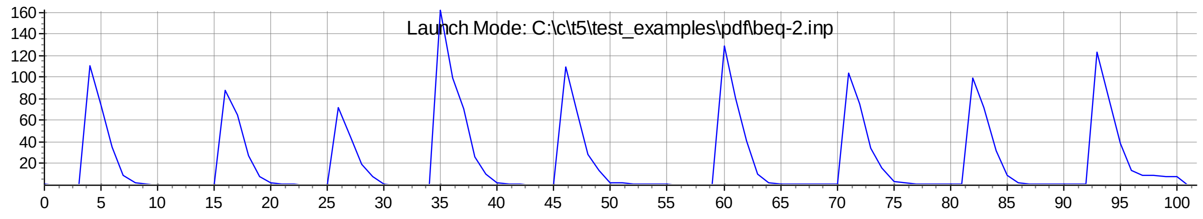
rebin_start_x_at can be used to place the start of the data at an integral multiple of the step size. In BEQ-2.INP parameters such as scale are written in terms of the rebin step size to reflect the fact that the scaling of the data is changed due to rebinning.

1.4.7 Refining on beq parameters

Modify the BB macro so that it comprises:

```
macro BB {   } ' beq, Insert/remove !
```

Gives an Rwp plot of:



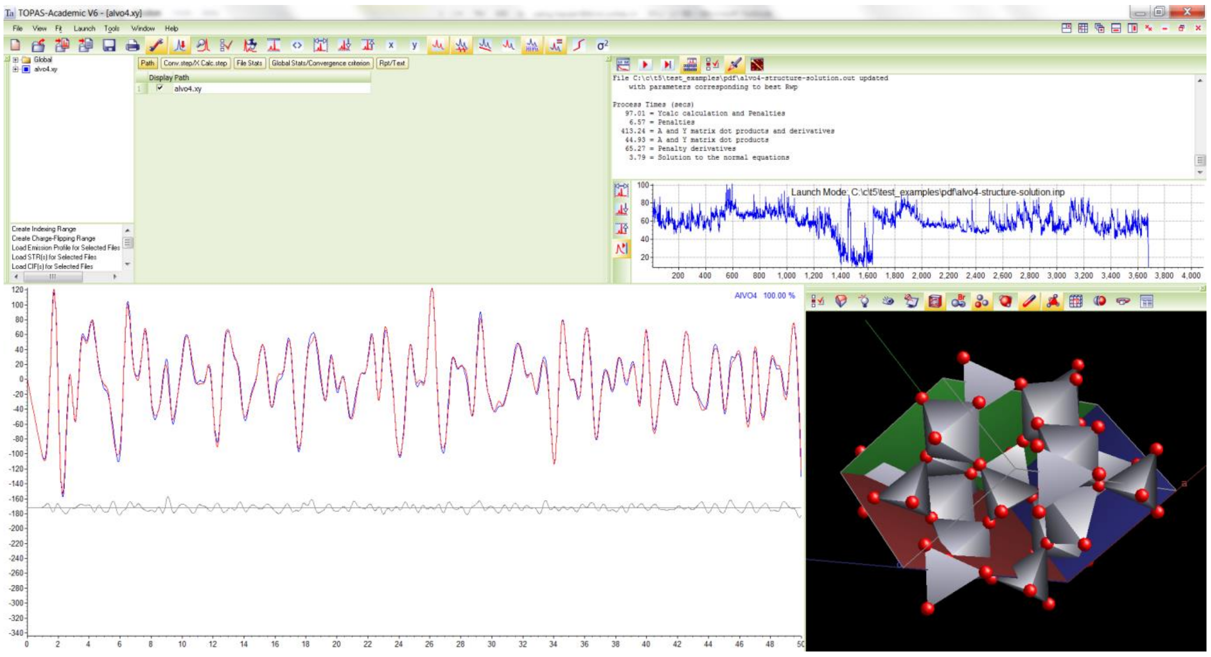
This type of convergence is indicative of derivatives being calculated correctly. Convergence for coordinates, occupancies, lattice parameters and pdf_zero are similar.

1.4.8 Structure Solution, Simulated Annealing

test_examples\pdf\alvo4\structure-solution-create.inp creates a simulated pattern for structure-solution.inp. It's a simulated annealing refinement with all coordinates starting at zero with anti-bump penalties applied using:

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

The correct solution is found as seen in the following:


```
File C:\c\t5\test_examples\pdf\alvo4-structure-solution.out updated
    with parameters corresponding to best Rwp

Process Times (secs)
   97.01 = Ycalc calculation and Penalties
    6.57 = Penalties
  413.24 = A and Y matrix dot products and derivatives
   44.93 = A and Y matrix dot products
   65.27 = Penalty derivatives
    3.79 = Solution to the normal equations
```
TOPAS-Academic V6 - [alvo4.xy]
File
View
Fit
Launch
Tools
Window
Help
Global
alvo4.xy
Path
Conv.step/X Calc.step
File Stats
Global Stats/Convergence criterion
Rpt/Text
Display Path
alvo4.xy
Create Indexing Range
Create Charge-Flipping Range
Load Emission Profile for Selected Files
Load STR(s) for Selected Files
Load CIF(s) for Selected Files
Launch Mode: C:\c\t5\test_examples\pdf\alvo4-structure-solution.inp
AlVO4 100.00 %

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

1.4.9 Rigid bodies with PDF data

test_examples\pdf\alvo4\rigid.inp operates on the simulated data created by structure-solution-create.inp. It demonstrates the use of rigid bodies with PDF data.

1.4.10 Occupancy merging with PDF data

test_examples\pdf\occ-merge-PbSO4\occ-merge.inp operates on simulated data created by create.inp. It demonstrates the use of occ_merge with PDF data.

1.4.11 Equivalence of pdf_gauss_fwhm and beq when there's one atom type

test_examples\pdf\si1.inp comprises an option to use *beq* or *pdf_gauss_fwhm* as follows:

For the beq case we have:

```
    beq = width;
```

and for pdf_gauss_fwhm we have:

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

The above cases are equivalent when there's one atom type.

2 NEW GUI FUNCTIONALITY

2.1 TOF x-axis can be displayed in d-spacing, Q and tof

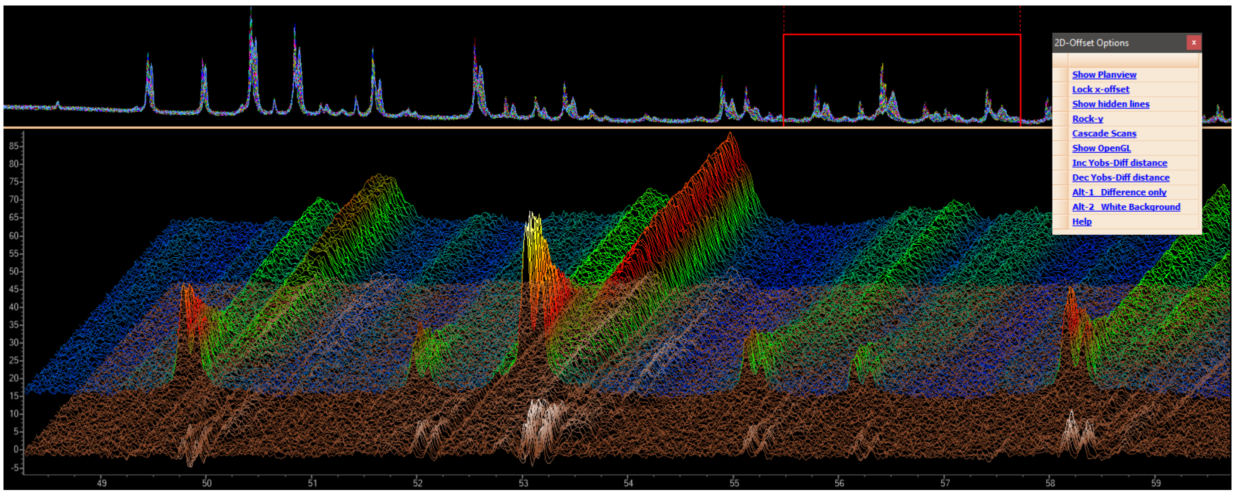
The x-axis of TOF data can be displayed as either tof, d-spacing or Q by cycling the x-axis button:

2.2 Displaying many files at once

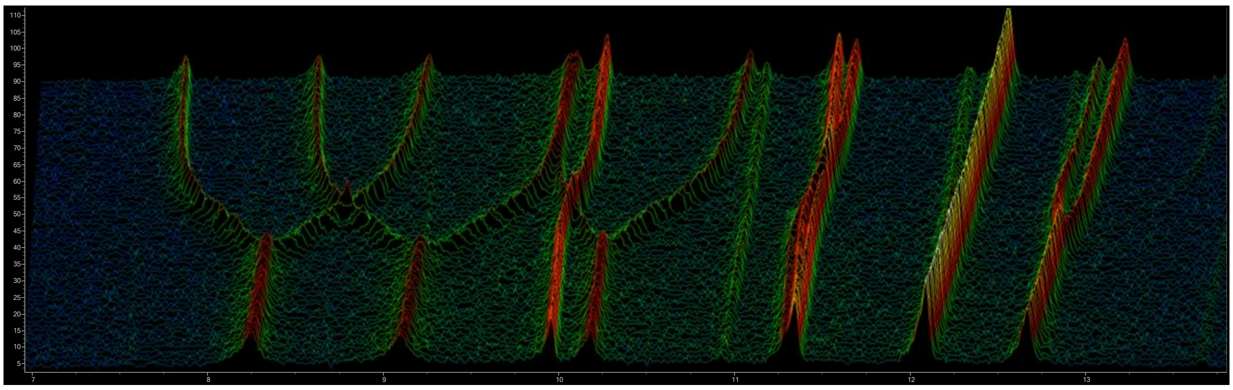
See files in the directory **test_examples\3d\**.

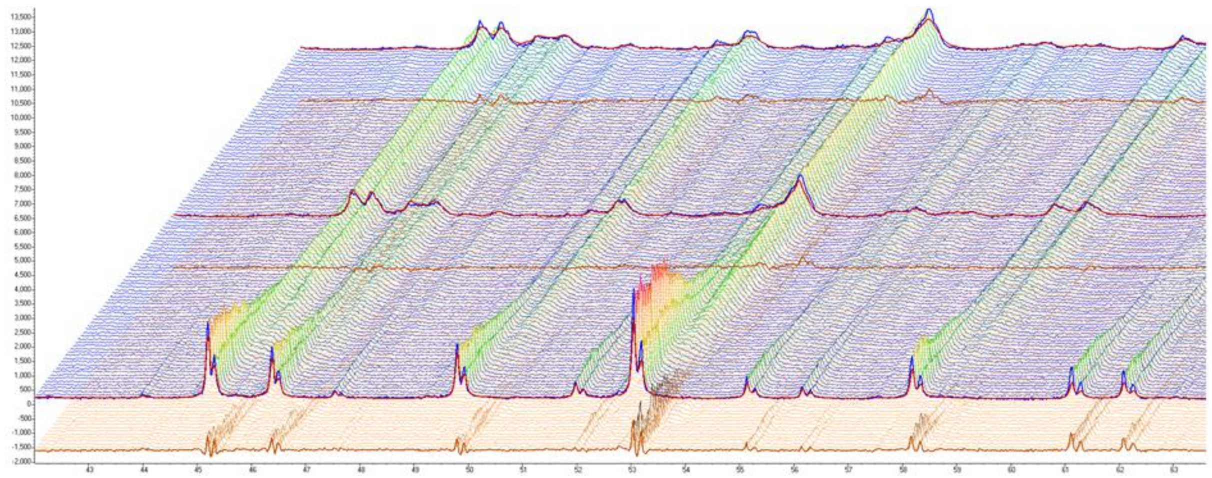
2.2.1 Surface plots – 2D with offsets

Scans can be displayed and offset from one another using the icon, for example:



The Quickzoom window is operational in all 2D-offset plots.

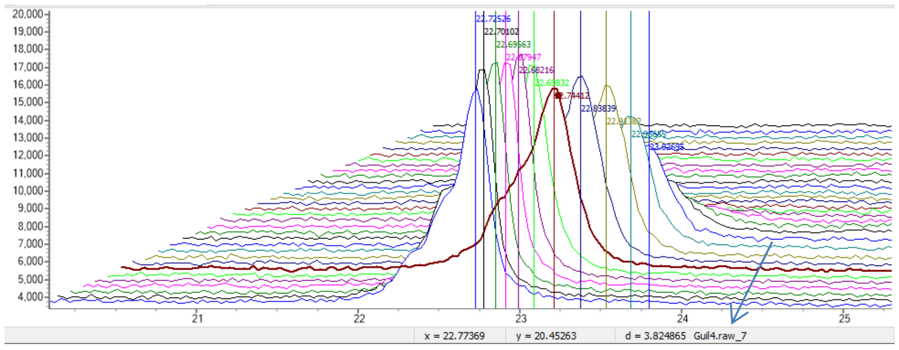




Pressing the **Middle Mouse Button** and moving the mouse changes the x and y offsets. This movement greatly assists in determining the curvature of the surface. The QuickZoom display is not offset allowing for two views of the same data.

2.2.2 Inserting peaks and identifying scans

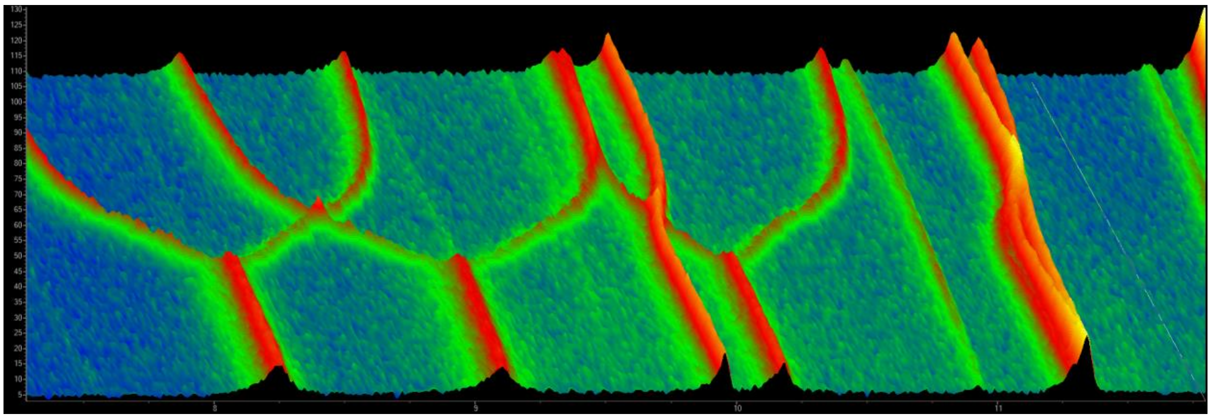
Peak can be inserted by pressing the **Ctrl-Key** and clicking the RMB. When the Ctrl key is pressed a solid circle is displayed on the scan closest to the mouse. The circle is coloured to match the scan lines and in addition the closest scan is displayed with a thickened line. Displayed at the bottom of the plot is the name of the scan as seen by the arrow below. Peaks as well as excluded regions move with the offsets.



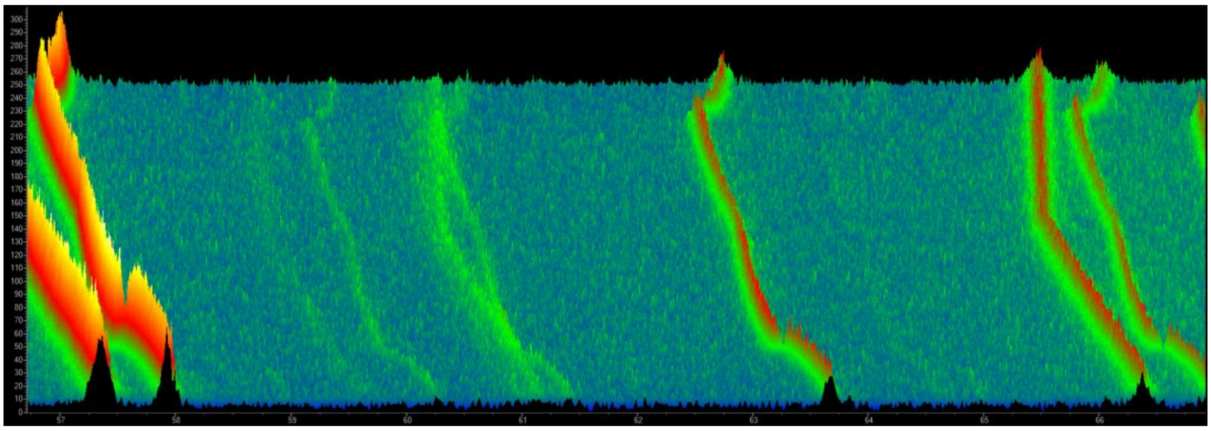
When the Ctrl-Key is pressed the x and y axis values displayed on the status line are offset to match the closest scan. Similarly when "For LAM Cursor" option is selected the LAM cursor is changed to match the axis of the closest scan.

2.2.3 2D-offset Surface plots

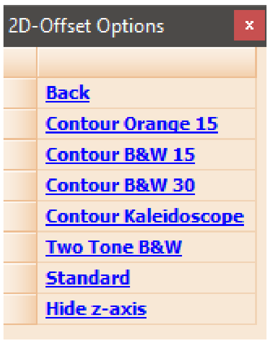
2D-offset plot can be displayed as a 3D-Surface, for example:



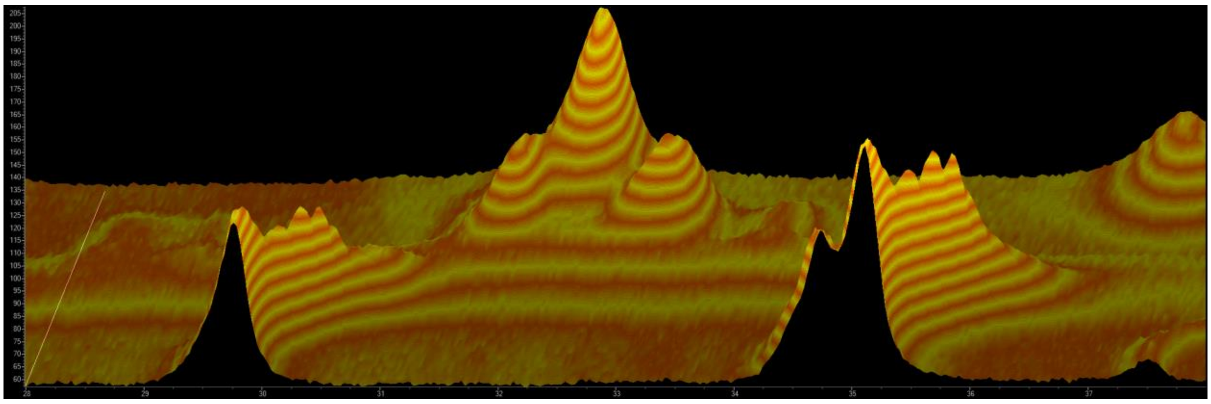
These plots can be manipulated in real time; the 871 file test_examples\je-para\d8_02999.raw with over 4 million data points can be easily manipulated:



Pressing the Shift key whilst performing a Zoom (forming a box using the mouse) zooms into a region. Zooming in this manner deselects scans for display. An unzoom is performed by performing an Unzoom whilst holding down the Shift key. Colour schemes can be changed by using the Colours options:

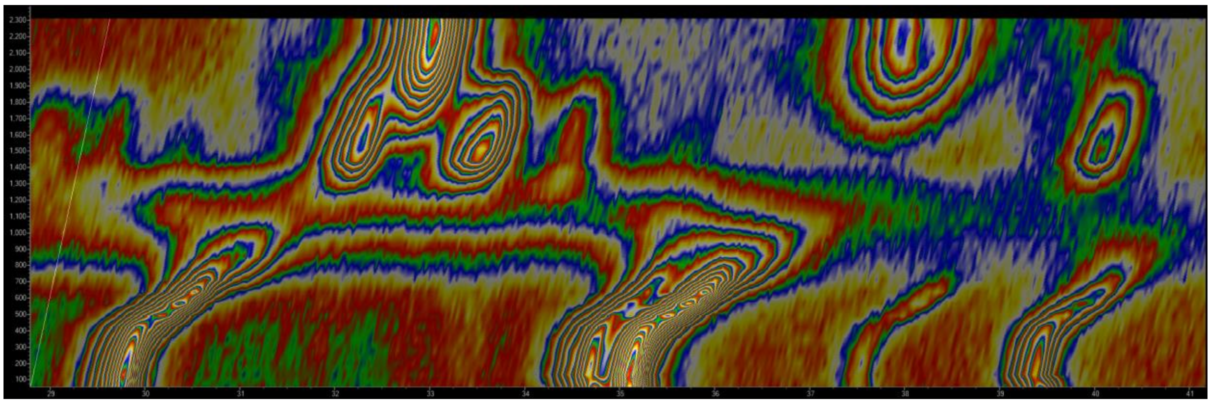


Contour-Orange-15 for looks like:

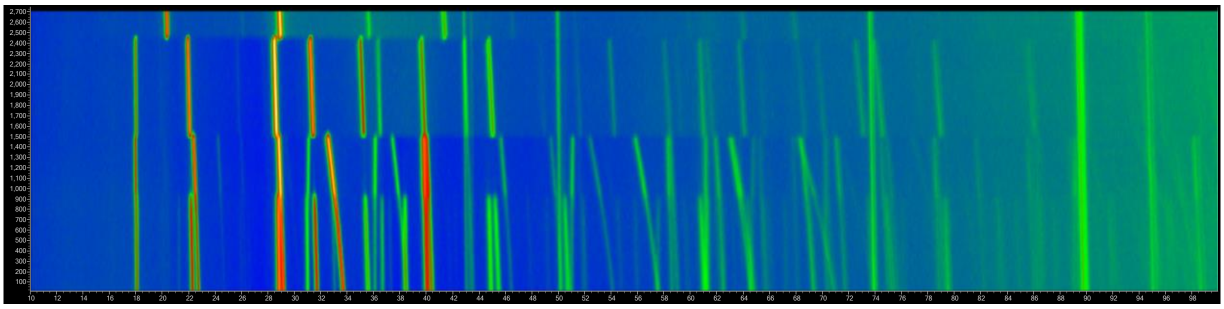


2.2.4 2D-offset Planview plots

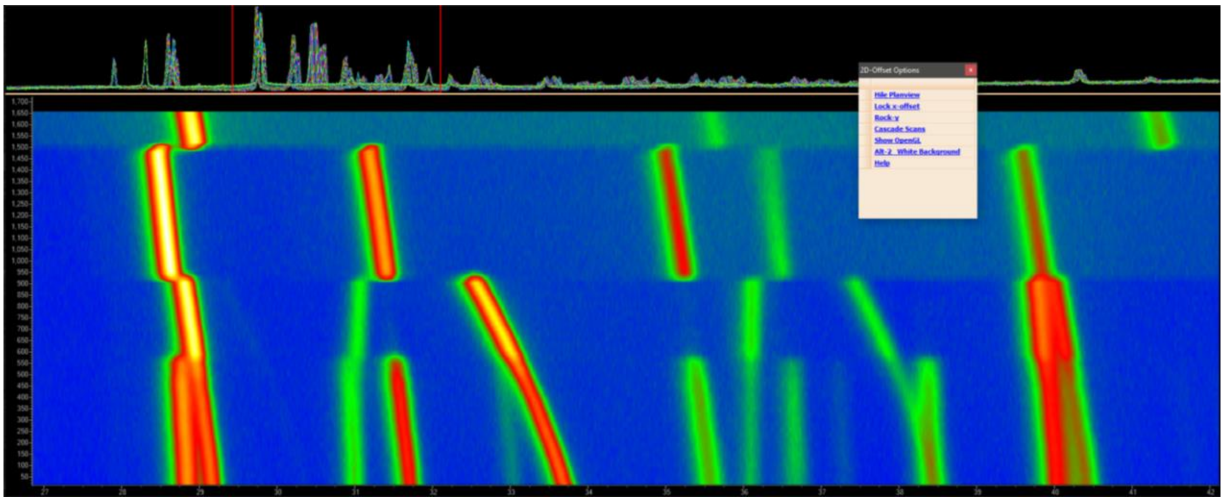
Moving the y-offset such that it's at a maximum automatically produces a Planview; a Kaleidoscope colour scheme gives:



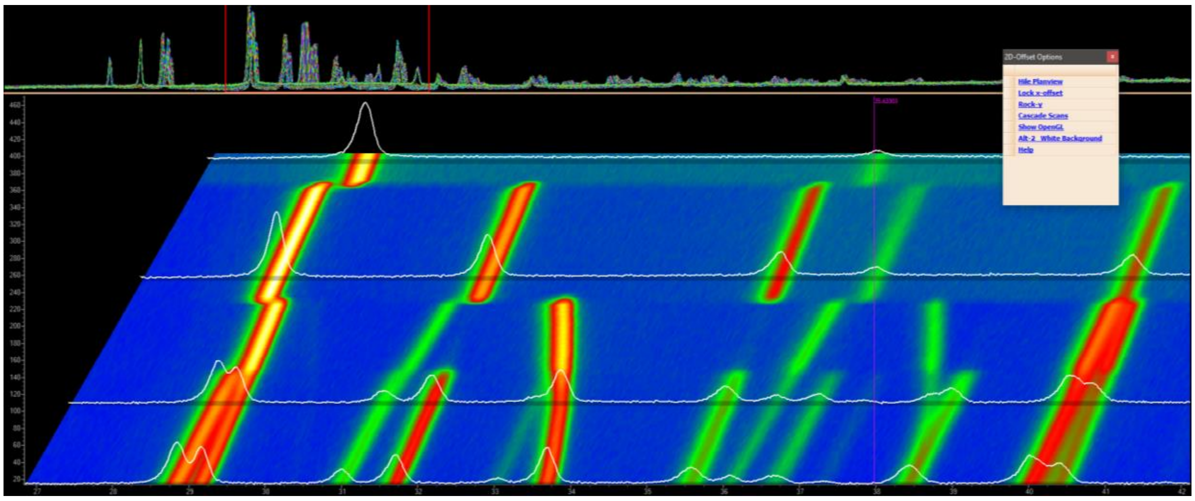
The Standard colour scheme gives:



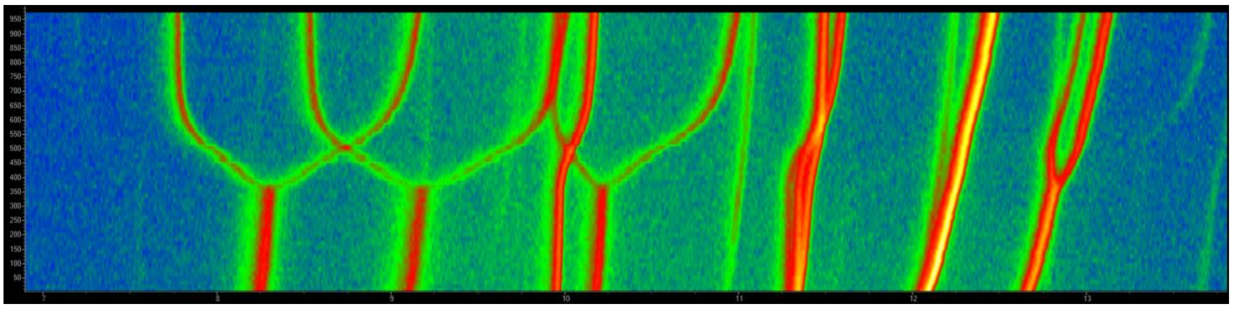
Zooming gives:



Planview can also have x-axis offsets with line scans overlain:

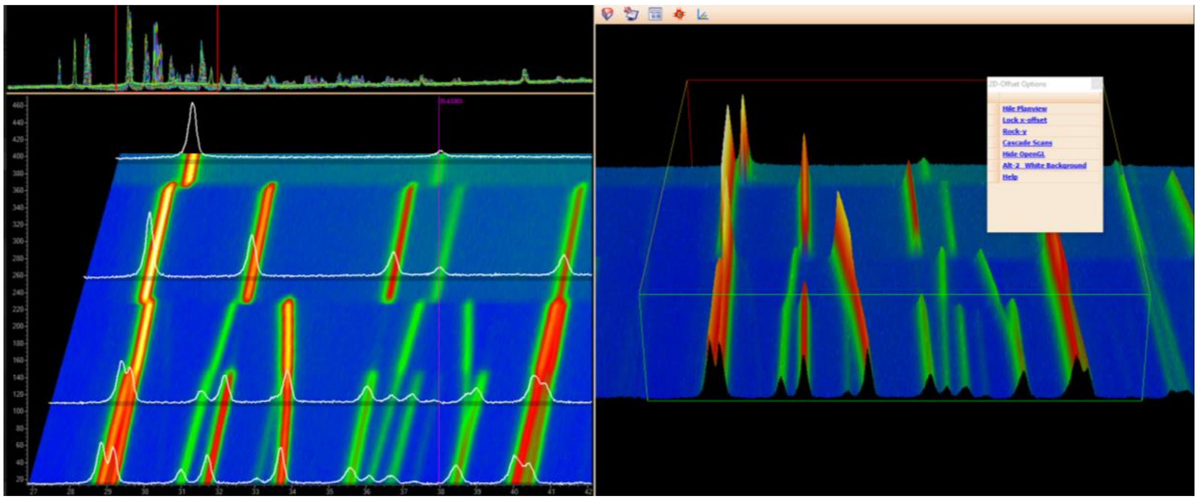


These line scans can include the calculated and/or difference patterns as well as patterns for individual phases. Beneath the displayed line scans are their shadows. Colours are blended across scans as well as across the x-axis to sharpen images.

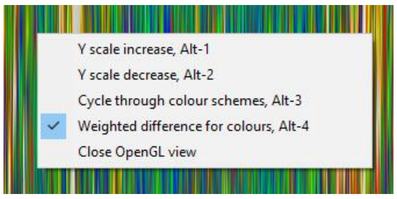


2.2.5 OpenGL Surface plots

OpenGL surface plots can be displayed alongside 2D-offset plots:



The scans displayed in the chart area are displayed to the right as a surface plot. Use RMB on the surface plot for options; these are:


Y scale increase, Alt-1
Y scale decrease, Alt-2
Cycle through colour schemes, Alt-3
✓ Weighted difference for colours, Alt-4
Close OpenGL view

The OpenGL surface plot respects the 2D x-axis and y-axis display options. It is also aware of the QuickZoom window and scrolling. Scrolling can be performed from either the 2D or 3D displays using the Mouse Wheel. Navigation in the OpenGL window is as follows:

- Use the Moise Wheel to scroll the x-axis from either the 2D or 3D plots.
- RMB-Pressed and moving zooms
- Pressing '**x**' whilst rotating allows rotation around an axis vertical to the screen.
- Pressing '**y**' whilst rotating allows rotation around an axis horizontal to the screen.
- Pressing '**z**' whilst rotating allows rotation around an axis perpendicular to the screen.
- Pressing the Mouse Wheel button (as opposed to rotating the mouse wheel) moves the object and hence the centre of rotation.
- When the Mouse is close to the Left or Right borders of the OpenGL window then rotation is around an axis perpendicular to the computer screen. Very useful for positioning 3D objects.

Opening the OpenGL Text Dialog and clicking on the 3D surface writes text into the Text Dialog; this text comprises the names of the two files bordering the polygon that has been clicked and the average x and y values of the polygon, for example:

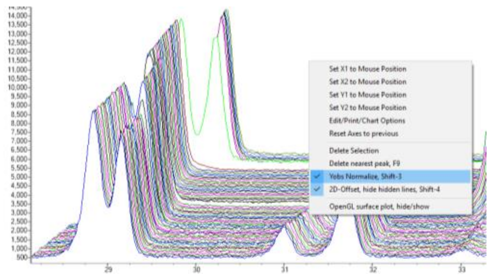
2.2.6 OpenGL – Weighted difference for colours

The RMB "Weight difference for colours" option displays colours corresponding to the

WtDiff = Abs(Yobs-Ycalc) / Weighting

2.3 Normalizing scans to the maximum scan value wiithin a Scan Window

Displayed scans can be normalized using the option “Yobs Normalize” which is activated using the RMB on the Scan window. Normalizing scales displayed scans such that the maximum values of the displayed data are all equal. Normalizing is temporary and can be toggled on/off by executing the “Yobs Normalize”. The following shows scans normalized with all the peaks on the right having the same height.



References

Treacy, M. M. J.; Newsam, J. M.; Deem, M. W. (1991) *Proceedings of the Royal Society-Mathematical and Physical Sciences*, **433**, 499.